

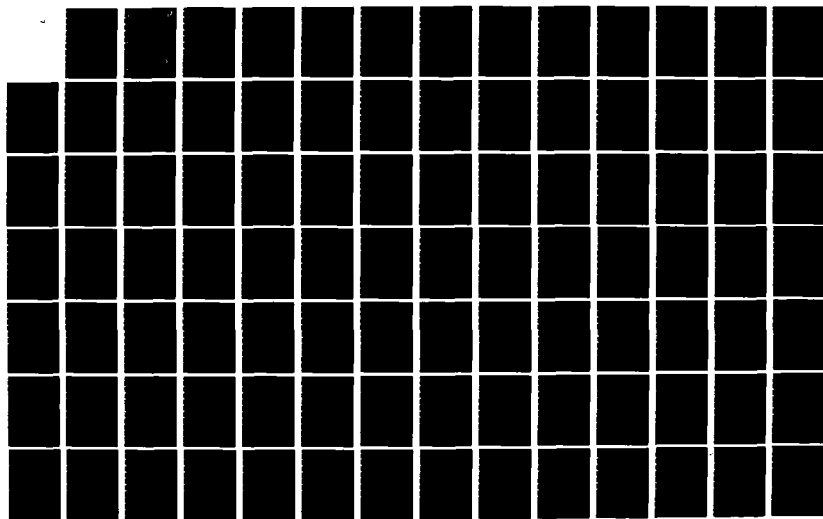
HD-A138 119

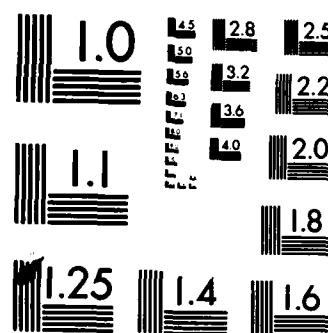
PROTOCOL STANDARDS AND IMPLEMENTATION WITHIN THE
DIGITAL ENGINEERING LABO. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. P W PHISTER
DEC 83 AFIT/GE/EE/83D-58 F/G 17/2

1/4

UNCLASSIFIED

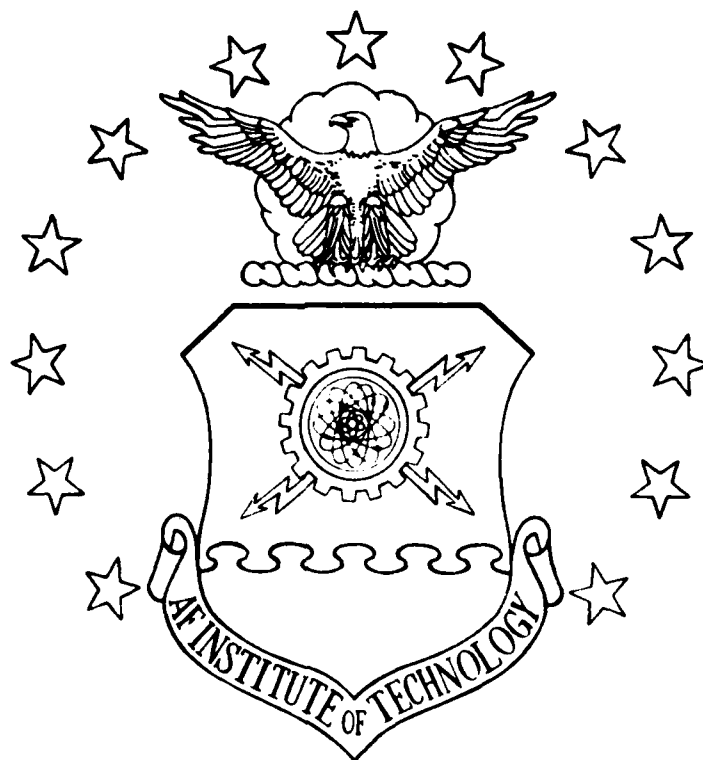
NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A138119



Protocol Standards and Implementation
Within the Digital Engineering Laboratory
Computer Network (DELNET)
Using the
Universal Network Interface Device
(UNID)

THESIS

(Part 2 of 2)

DTIC
ELECTRONIC
S FEB 22 1984
D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC FILE COPY

84 02 21 198

AFIT/GE/EE/83D-58

Accession For	
NTIS	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/1	



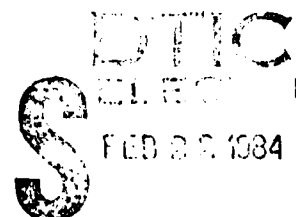
Protocol Standards and Implementation
Within the Digital Engineering Laboratory
Computer Network (DELNET)
Using the
Universal Network Interface Device
(UNID)

THESIS

(Part 2 of 2)

AFIT/GE/EE/83D-58

Paul W. Phister, Jr.
Capt USAF



Approved for public release; distribution unlimited.

AFIT/GE/EE/83D-58

Protocol Standards and Implementation
Within the Digital Engineering Laboratory
Computer Network (DELNET)
Using the
Universal Network Interface Device
(UNID)

THESIS

(Part 2 of 2)

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Paul W. Phister, Jr., BS.EE, MS.SM

Capt

USAF

Graduate Electrical Engineering

December 1983

Approved for public release; distribution unlimited.

APPENDIX E
Data Dictionary

Introduction

This appendix contains the data dictionary for the fifteen modules that comprise the DELNET up to the end of this investigation. Two modules compose the UNIDS startup software; Eight modules compose the UNIDS Local and Network Operating Systems; and lastly, five modules compose the Zilog Operating System.

This data dictionary is organized by modules which are presented in alphabetical order. Each module contains a section for constants, variables, and procedures which are in turn listed in alphabetical order. The last section of this appendix contains a cross reference list for all constants, variables, and procedures along with the modules in which they are located.

As mentioned throughout this investigation effort, each software module may vary depending on which UNID is being referenced. Currently, the only difference between UNID#1 and UNID#2 software is the variable THIS_UNID_NBR along with some textual difference. As the DELNET progresses, the UNIDS software will be more and more tailored to the Local Area Networks it services. An example would be the data transfer speed between Local Area Network and a particular UNID Local Channel.

Table of Contents

This appendix is subdivided into five sections, which are:

<u>Modules</u>		<u>Page</u>
1. UNID Start-up Modules.....		E-3
UNID#1	UNID#2	
UNID_INIT_ONE	UNID_INIT_TWO.....	E-3
UNID_ONE	UNID_TWO.....	E-4
2. UNID Operating System.....		E-5
UNID#1	UNID#2	
L.MAIN_U1	L.MAIN_U2.....	E-5
L.TAB_U1	L.TAB_U2.....	E-9
L.VINT_U1	L.VINT_U2.....	E-11
N.INSIO_U1	N.INSIO_U2.....	E-12
N.MAIN_U1	N.MAIN_U2.....	E-14
N.TAB_U1	N.TAB_U2.....	E-18
U.LIB_U1	U.LIB_U2.....	E-20
U.SHTAB_U1	U.SHTAB_U2.....	E-21
3. ZILOG Operating System.....		E-23
ZILOG_APPLICATION.....		E-23
ZILOG_LIB.....		E-26
ZILOG_PHYSICAL.....		E-27
ZILOG_TABLES.....		E-28
ZILOG_TRANSPORT.....		E-30
4. Data Dictionary Cross Reference.....		E-37
5. Appendix Summary.....		E-44

1. UNID Start-up Modules:

MODULES UNID_INIT_ONE UNID_INIT_TWO

The purpose of these assembly language modules is to initially start-up the UNIDs. UNID_INIT_ONE starts up UNID#1 and UNID_INIT_TWO starts up UNID#2.

Constants

CTC0 - Internal, type hexadecimal - CTC-1 Channel-0 Port Address.

CTC1 - Internal, type hexadecimal - CTC-1 Channel-1 Port Address.

INTVEC - Internal, type hexadecimal - Interrupt Page Address.

TRANSFER - Internal, type hexadecimal - Main program starting address.

USART - Internal, type hexadecimal - USART Port Address.

VECADD- Internal, type hexadecimal- InterruptTable Address.

Variables

None.

Procedures

UNIDIN - Initializes the USART and CTC.

MODULES UNID_ONE UNID_TWO

The purpose of these assembly language programs is to transfer the Local and Network Operating Systems from the ZILOG MCZ 1/25 Computer System to the UNIDs. UNID_ONE sends the programs to UNID#1 and UNID_TWO sends the programs to UNID_TWO.

Constants

CTC1 - Internal, type hexadecimal - CTC-1 Port Address.

RCVRDY - Internal, type hexadecimal - Receive Ready Bit.

USART - Internal, type hexadecimal - USART Port Address.

Variables

None.

Procedures

Transfer - Transfers the Local and Network Operating System files from the ZILOG MCZ to the UNID.

2. UNID Operating System:

MODULES L.MAIN_U1 L.MAIN_U2

The purpose of these PLZ language modules is to provide the UNID Local Operating System with the main line of processing. The Local Operating System will Input/Output data from the four local channels or transfer and receive data from the Network Operating System of the UNID. Module L.MAIN_U1 is used in UNID#1 and module L.MAIN_U2 is used in UNID#2.

Constants

CONCMD - Command port address for the USART on the local monitor console.

CONCTC - CTC port address for the USART on the local monitor console.

CONDAT - Dataport address for the USART on the local monitor console.

DATA_SIZE - Number of bytes received by a Local Host.

DATA_TABLE_SIZE - Number of bytes within a data table.

L_RI_DEST_ERR - Local route in destination error.

L_RO_DEST_ERR - Local route out destination error.

MAX_COUNTRY_CODE - Maximum number of Countries operational on the DELNET.

MAX_NETWORK_CODE - Maximum number of UNIDs operational within a particular Country.

PACKET_SIZE - Number of bytes in a packet.

PACKETS_IN_TABLE - Number of packets in a packet table.

PACKET_TABLE_SIZE - Number of bytes in a packet table.

STAT_NBR - Number of the status entries to be included in the status table.

THIS_COUNTRY_CODE - Unique code indicating which Country THIS_UNID_NBR resides.

THIS_UNID_NBR - Unique UNID number for the UNID performing the interface between Local Hosts and the DELNET.

U01DAT - Local Channel-1 USART data port address.

U02DAT - Local Channel-2 USART data port address.

U03DAT - Local Channel-3 USART data port address.

U04DAT - Local Channel-4 USART data port address.

Variables

BITS- Local, type integer - Integer value of
Control_Code, Country_Code, or Network_Code.

CONTROL_CODE - Local, type byte - Indicates the routing
format used in the Destination Address field of
the Internet-Header-Format Header.

COUNTRY_CODE - Local, type byte - Area where the dest.
UNID resides.

DESTINATION - Internal, type word - The destination
address of either the 128-byte data block or
data packet.

DESTINATION_ADDRESS - Local, type byte - The destination
address for the data packet.

DESTINATION_BYTE - Local, type byte - Temporary location
for the destination address used in
BUILD_I_PACKET.

HOST_CODE - Local, type integer - Number indicating
which local host the data is to be sent to.

IX - Local, type integer - An index for iteration loops.

NETWORK_CODE - Local, type byte - UNID destination.

PRTADD - Local, type byte - Contains the number of bytes
that will be sent from one location to another.

SEQUENCE_BYTE - Local, type byte - Sequence control
section within the Frame Header.

SOURCE_ADDRESS - Local, type byte - The source address
of the incoming data.

SOURCE_BYTE - Local, type byte - Temporary location for
the source address used in BUILD_I_PACKET.

SPARE_01 - Local, type byte - Not implemented.

SPARE_02 - Local, type byte - Not implemented.

SRCADD - Local, type Pbyte - Contains the address of
the data location that requires servicing.

STARTUP_HDR - Internal, type array - A message to the local monitor console indicating proper operating system operation.

TABLE - Internal, type word - Used as a pointer to indicate which table is to be serviced.

TDAADD - Global, type Pbyte - Starting address of data to be transmitted out the USARTs.

TPRADD - Global, type byte - Data port address for the local USARTs.

Procedures

BUILD_I_PACKET - A procedure which transforms a 128-byte data block into a 133-byte data packet.

DET_DEST_FOUR - A procedure that determines the destination of a 128-byte data block coming into Local Channel-4.

DET_DEST_LL - A procedure that determines the dest. of data located in the Local-to-Local Table.

DET_DEST_LN - A procedure that determines the dest. of data located in the Local-to-Network Table.

DET_DEST_ONE - A procedure that determines the dest. of a 128-byte data block coming into Local Channel-1.

DET_DEST_THREE - A procedure that determines the dest. of a 128-byte data block coming into Local Channel-3.

DET_DEST_TWO - A procedure that determines the dest. of a 128-byte data block coming into Local Channel-2.

LD_TAB_HSKP - Load Table Housekeep - Housekeeps a specified table after a new 128-byte data block or a data packet has been loaded.

MAIN - This is the main procedure which drives all the other procedures through their proper sequencing.

ROUTE_IN - Routes in 128-byte data blocks from their correct input tables and places them into their correct output tables for transmit.

ROUTE_OUT - Routes out either 128-byte data blocks or data packets from their correct output tables to their correct output channels.

SRVC_TAB_HSKP - Service Table Housekeep - Housekeeps a specified table whenever a 128-byte data block or a data packet is removed from the table.

TRNMIT_PKT - Transmits a 128-byte data block out of one of the two output tables to one of the local channels.

MODULES L.TAB_U1 L.TAB_U2

The purpose of these PLZ modules is to provide the Local Operating System with the tables necessary for storing the 128-byte data blocks after reception and before transmission to the local hosts through one of the four local channels. Module L.TAB_U1 is used in UNID#1 and module L.TAB_U2 is used in UNID#2.

Constants

DATA_SIZE - Number of bytes of data in an input data table.

DATA_TABLE_SIZE - Number of data bytes within a data table.

PACKETS_IN_TABLE - Number of data packets in a packet table.

Variables

LC01NE - Global, type integer - Pointer for the next available position within the table LC01TB.

LC01NS - Global, type integer - Pointer for the next byte to be serviced within the table LC01TB.

LC01SZ - Global, type integer - Size of the LC01TB table.

LC01TB - Global, type array - Local input table which interfaces with local channel-1.

LC02NE - Global, type integer - Pointer for the next available position within the table LC02TB.

LC02NS - Global, type integer - Pointer for the next byte to be serviced within the table LC02TB.

LC02SZ - Global, type integer - Size of the LC02TB table.

LC02TB - Global, type array - Local input table which interfaces with local channel-2.

LC03NE - Global, type integer - Pointer for the next available position within the table LC03TB.

LC03NS - Global, type integer - Pointer for the next byte to be serviced within the table LC03TB.

LC03SZ - Global, type integer - Size of the LC03TB table.

LC03TB - Global, type array - Local input table which interfaces with local channel-3.

LC04NE - Global, type integer - Pointer for the next available position within the table LC04TB.

LC04NS - Global, type integer - Pointer for the next byte to be serviced within the table LC04TB.

LC04SZ - Global, type integer - Size of the LC04TB table.

LC04TB - Global, type array - Local input table which interfaces with local channel-4.

LCLCNE - Global, type integer - Pointer for the next available byte position within the table LCLCTB.

LCLCNS - Global, type integer - Pointer for the next byte to be serviced within the table LCLCTB.

LCLCSZ - Global, type integer - Size of the LCLCTB table.

LCLCTB - Global, type array - Local-to-Local table that receives the 128-byte data blocks that are destined for other local hosts on one of the four local channels.

Procedures

INIT_L_TAB - Initilize Local Tables - Sets up the local tables and initilizes the pointers to zero.

MODULES L.VINT_U1 L.VINT_U2

The purpose of these assembly language modules is to support the UNIDS Local Operating System and its processing. Module L.VINT_U1 is used in UNID#1 and L.VINT_U2 is used in UNID#2.

Constants

PICADD - PIC Port Address.

PICCMD- Control to allow any interrupt with no priority assignment.

Variables

None.

Procedures

INVINT - Initilize Vector Interrupt Mode - The purpose of this procedure initializes the vector interrupt process through the use of the Priority Interrupt Controller (PIC).

INIURT - Initialize Local Card USARTs - Initilizes the 2651 USARTs on the UNIDS Local Card.

TRNMIT - Transmit - The purpose of this procedure is to enable a transmit interrupt from a PLZ module.

URTR01 - I/O Receive Interrupt Controller-1 - The purpose of this procedure is to service local channel-1 receive interrupts.

URTR02 - I/O Receive Interrupt Controller-2 - The purpose of this procedure is to service local channel-2 receive interrupts.

URTR03 - I/O Receive Interrupt Controller-3 - The purpose of this procedure is to service local channel-3 receive interrupts.

URTR04 - I/O Receive Interrupt Controller-4 - The purpose of this procedure is to service local channel-4 receive interrupts.

URTRN - I/O Transmit Interrupt - The purpose of this procedure is to service the local channel transmit interrupts.

MODULES N.INSIO_U1 N.INSIO_U2

The purpose of these assembly language modules is to support the network operating system of the UNID and its processing of data packets and frames.

Constants

- A_DATA - Internal, type hexadecimal - SIO Channel-A Data Address.
- A_STAT - Internal, type hexadecimal - SIO Channel-A Status/Command Address.
- B_DATA - Internal, type hexadecimal - SIO Channel-B Data Address.
- B_STAT - Internal, type hexadecimal - SIO Channel-B Status/Command Address.
- CTC_0 - Internal, type hexadecimal - CTC Channel-0 Address.
- CTC_1 - Internal, type hexadecimal - CTC Channel-1 Address.
- CTC_2 - Internal, type hexadecimal - CTC Channel-2 Address.
- CTC_3 - Internal, type hexadecimal - CTC Channel-3 Address.
- FALSE - Internal, type hexadecimal - True or false flag.
- TC1 - Internal, type hexadecimal - Time constant for Channel-1.
- TC2 - Internal, type hexadecimal - Time constant for Channel-2.
- TC3 - Internal, type hexadecimal - Time constant for Channel-3.

Variables

None.

Procedures

- INSIO - Initilize SIO - The purpose of this procedure is to initilize the I/O process for frames transmitted and received on both Channel-A and Channel-B.

- RECCHA - Receive Channel-A - This procedure services the receive interrupt requests for frames coming into the UNID from network Channel-A.
- RECCHB - Receive Channel-B - This procedure services the receive interrupt requests for frames coming into the UNID from network Channel-B.
- STCTC2 - Start CTC Channel-2 - This procedure sets up the CTC Channel-2 for proper operation. STCTC2 is used for timeout on network Channel-A.
- STCTC3 - Start CTC Channel-3 - This procedure sets up the CTC Channel-3 for proper operation. STCTC3 is used for timeout on network Channel-B.
- XMITCA - Transmit Channel-A - This procedure transmits a frame from network Channel-A onto the network.
- XMITCB - Transmit Channel-B - This procedure transmits a frame from network Channel-B onto the network.

MODULES N.MAIN_U1 N.MAIN_U2

The purpose of these PLZ language modules is to provide the UNID Network Operating System with the main line of processing. The network operating system will input/output data from either network channels A or B or transfer and receive data packets from the local operating system of the UNID.

Constants

CONCMD - Command port address for the USART on the network monitor console.

CONCTC - CTC port address for the USART on the network monitor console.

CONDAT - Data port address for the USART on the network monitor console.

FALSE - Boolean word used for high order control.

FRAME_TABLE_SIZE - Number of bytes in a frame table.

FRAMES_IN_TABLE - Number of frames in a frame table.

FRAME_SIZE - Number of bytes in a frame.

MAX_COUNTRY_CODE - Indicates the maximum countries currently operational within the DELNET.

MAX_NETWORK_CODE - Indicates the number of UNIDs operational within a particular country.

NET_RI_DEST_ERR - Network route in destination error.

NET_RO_DEST_ERR - Network route out destination error.

PACKETS_IN_TABLE - Number of packets in a packet table.

PACKET_SIZE - Number of bytes in a data packet.

STAT_NBR - Number of status entries to be included in the status table.

THIS_COUNTRY_CODE - Country where THIS_UNID_NBR resides.

THIS_UNID_NBR - Unique number for this UNID. This number must reflect the correct UNID or incorrect routing will occur.

TRUE - Opposite of False.

Variables

- ACKNOWLEDGE_A - Internal, type byte - Indicates either true or false if a good acknowledgement frame has been received from network channel-A.
- ACKNOWLEDGE_B - Internal, type byte - Indicates either true or false if a good acknowledgement frame has been received from network channel-B.
- ADDRESS_BYTE - Internal, type byte - Address field within the Frame Header.
- ADDRESS_WORD - Internal, type byte - Address field within an S-Frame Header.
- CONTROL_BYTE - Internal, type byte - Control field within the Frame Header.
- CONTROL_WORD - Internal, type byte - Address field within an S-Frame Header.
- CTCNOA - Global, type byte - Counter for the CTC. Incremented once each timeout of the CTC for network channel-A.
- CTCNOB - Global, type byte - Counter for the CTC. Incremented once each timeout of the CTC for network channel-B.
- DESTINATION - Internal, type word - Destination of a data packet or a frame.
- DESTINATION_UNID - Internal, type integer - Number that indicates the destination UNID for the Frame.
- DISTANCE_LEFT - Internal, type integer - Number indicating the number of hops left to the destination UNID.
- DISTANCE_RIGHT - Internal, type integer - Number indicating the number of hops right to the destination UNID.
- INPUT_SEQ_BIT - Internal, type byte - Sequence bit (Mod-2) to be entered into a new frame.
- MAXNOA - Global, type byte - The maximum number of times the CTC will cycle through its counting routine for network channel-A.
- MAXNOB - Global, type byte - The maximum number of times the CTC will cycle through its counting routine for network channel-B.

SEQ_BIT_A - Internal, type byte - Sequence bit (Mod-2) of an active I-Frame for network channel-A.

SEQ_BIT_B - Internal, type byte - Sequence bit (Mod-2) of an active I-Frame for network channel-B.

STARTUP_HDR - Internal, type array - A message to the network monitor indicating proper operating system operation.

TABLE - Internal, type word - Used as a pointer to indicate which table is to be serviced.

THIS_SEQ_BIT - Internal, type byte - Sequence bit that is presently under examination.

TIMCHA - Global, type byte - Indicates either true or false if the TIME_DELAY_CHA procedure is complete.

TIMCHB - Global, type byte - Indicates either true or false if the TIME_DELAY_CHB procedure is complete.

Procedures

BUILD_I_FRAME - A procedure which builds an I-frame and places it in output table for network transmission.

BUILD_S_FRAME - A procedure which builds an S-frame and places it in output table for network transmission.

DET_DEST_LN - A procedure which determines the destination network channel of a data packet in table LCNTTB.

DET_DEST_ONE - A procedure which determines the destination of a frame located in table NT01TB.

DET_DEST_TWO - A procedure which determines the destination of a frame located in table NT02TB.

LD_TAB_HSKP - Load Table Housekeep - Housekeeps a specified table after a new packet or frame has been loaded.

MAIN - This is the main procedure which drives all the other procedures through their proper sequencing.

ROUTE_IN - A procedure that routes in frames from both of the network channels and places them into their correct tables for evaluation.

ROUTE_OUT - A procedure that routes out frames from either the local or network side of the UNID.

SRVC_TAB_HSKP - Service Table Housekeep - Housekeeps a specified table whenever a data packet or frame is removed.

TIME_DELAY_CHA - Creates a time delay between successive transmissions of I-frames on the Network Channel-A.

TIME_DELAY_CHB - Creates a time delay between successive transmissions of I-frames on the Network Channel-B.

MODULES N.TAB_U1 N.TAB_U2

The purpose of these PLZ language modules is to provide the network operating system with the tables necessary for storing the frames of data after reception and before transmission to the network bus. Module N.TAB_U1 is used in UNID#1 and module N.TAB_U2 is used in UNID#2.

Constants

FRAME_SIZE - Number of bytes in a frame.

FRAMES_IN_TABLE - Number of frames in a frame table.

FRAME_TABLE_SIZE - Number of bytes in a frame table.

PACKETS_IN_TABLE - Number of packets in a packet table.

PACKET_SIZE - Number of bytes in a data packet.

Variables

NT01NE - Global, type integer - Table pointer for the next available position within the table NT01TB.

NT01NS - Global, type integer - Table pointer for the next byte to be serviced within the table NT01TB.

NT01SZ - Global, type integer - Size of the table NT01TB.

NT01TB - Global, type array - Network Channel-A input table from the network.

NT02NE - Global, type integer - Table pointer for the next available position within the table NT02TB.

NT02NS - Global, type integer - Table pointer for the next byte to be serviced within the table NT02TB.

NT02SZ - Global, type integer - Size of the table NT02TB.

NT02TB - Global, type array - Network Channel-B input table from the network.

OUTFRAME_CHA_NE - Global, type integer - Table pointer for the next available position within the table OUTFRAME_CHA_TB.

OUTFRAME_CHA_NS - Global, type integer - Table pointer for the next byte to be serviced within the table OUTFRAME_CHA_TB.

OUTFRAME_CHA_SZ - Global, type integer - Size of the table OUTFRAME_CHA_TB.

OUTFRAME_CHA_TB - Global, type array - Network Channel-A output table going to the network.

OUTFRAME_CHB_NE - Global, type integer - Table pointer for the next available position within the table OUTFRAME_CHB_TB.

OUTFRAME_CHB_NS - Global, type integer - Table pointer for the next byte to be serviced within the table OUTFRAME_CHB_TB.

OUTFRAME_CHB_SZ - Global, type integer - Size of the table OUTFRAME_CHB_TB.

OUTFRAME_CHB_TB - Global, type array - Network Channel-B output table going to the network.

Procedures

INIT_N_TAB - Initilize the Network Tables - Sets up the network tables and initializes the data pointers to zero.

MODULES U.LIB_U1 U.LIB_U2

The purpose of these assembly language modules is to support both the local and network operating systems of the UNID with a series of library routines.

Constants

None.

Variables

None.

Procedures

MOVSEQ - Move Sequence - A procedure to move a block of data from one area of memory to another.

RECSEQ - Receive Sequence - A procedure that receives a sequence of bytes from an identified port.

SNDSEQ - Send Sequence - A procedure that sends a block of data out an identified port.

MODULES U.SHTAB_U1 U.SHTAB_U2

The purpose of these PLZ language modules is to provide both the local and network operating systems of the UNID with a shared interface for which they can exchange information. In the present form this interface is a pair of tables which will be located in the shared memory partition of the UNID memory. Module U.SHTAB_U1 is used in UNID#1 and module U.SHTAB_U2 is used in UNID#2.

Constants

DATA_SIZE - Size of the data block from the Host.

PACKET_TABLE_SIZE - Number of bytes in a packet table.

PACKETS_IN_TABLE - Number of data packets in a packet table.

PACKET_SIZE - Number of bytes in a data packet.

STAT_NBR - Number of status entries to be included in the status table.

Variables

LCNTNE - Global, type integer - Data pointer for the next available position within the table LCNTTB.

LCNTNS - Global, type integer - Data pointer for the next byte to be serviced within the table LCNTTB.

LCNTSZ - Global, type integer - Size of the table LCNTTB.

LCNTTB - Global, type array - Table used for storing packets received from the local side destined for the network side of the UNID.

NTLCNE - Global, type integer - Data pointer for the next available position within the table NTLCTB.

NTLCNS - Global, type integer - Data pointer for the next byte to be serviced within the table NTLCTB.

NTLCSZ - Global, type integer - Size of the table NTLCTB.

NTLCTB - Global, type array - Table used for storing frames received from the network side destined for the local side of the UNID.

STATTB - Global, type array - Table used to keep track of local and network errors.

Procedures

INIT_U_SHTAB - Initialize Shared Tables - Initializes the tables shared by both the Local and Network Operating Systems and sets the data pointers to zeros.

3. Zilog Operating System:

MODULE ZILOG_APPLICATION

The purpose of this PLZ module is to provide the Zilog MCZ 1/25 Computer System with the Application Layer Software.

Constants

DATA_SIZE - Number of bytes rcvd/xmitted to the UNID.

DATA_TABLE - Number of messages that will be buffered.

DATA_TABLE_SIZE - Number of bytes of data in a data table.

ERROR_TABLE_SIZE - Number of bytes in the Error Table.

H19_COMMAND_PORT - Command port address for the H19 Terminal.

H19_DATA_PORT - Data port address for the H19 Terminal.

J112_DATA_PORT - Data port for J-112 of the Zilog.

J113_DATA_PORT - Data port for J-113 of the Zilog.

PRINTER_COMMAND_PORT - Command port for the Spinwriter.

PRINTER_DATA_PORT - Data port for the Spinwriter.

Variables

CONTROL_CODE - Local, type byte - Indicates the routing format used in the Destination Address field of the Internet Header Format portion of the Datagram.

DATAGRAM_LENGTH - Local, type byte - Datagram header length.

DESTINATION - Local, type word - Used to indicate correct routing. Two possible values are 'OK' and 'ER'.

DESTINATION_ADDRESS_FOUR - Global, type byte - The last 8-bits of the Destination Address in the Datagram.

DESTINATION_ADDRESS_ONE - Global, type byte - The first 8-bits of the Destination Address in the Datagram.

DESTINATION_ADDRESS_THREE - Global, type byte - The third 8-bits of the Destination Address in the Datagram.

DESTINATION_ADDRESS_TWO - Global, type byte - The second 8-bits of the Destination Address in the Datagram.

IX - Local, type integer - Used as a pointer.

PORT_ADDRESS - Local, type byte - The Datagram Port Address.

PORT_CODE_HIGH - Local, type byte - The upper four bits of the Port Code within the Destination Address of the Datagram.

PORT_CODE_LOW - Local, type byte - The lower four bits of the Port Code within the Destination Address of the Datagram.

SOURCE_ADDRESS_FOUR - Global, type byte - The last 8-bits of the Source Address in the Datagram.

SOURCE_ADDRESS_ONE - Global, type byte - The first 8-bits of the Source Address in the Datagram.

SOURCE_ADDRESS_THREE - Global, type byte - The third 8-bits of the Source Address in the Datagram.

SOURCE_ADDRESS_TWO - Global, type byte - The second 8-bits of the Source Address in the Datagram.

TABLE - Internal, type word - Used as a pointer to indicate which table is to be serviced.

TEST_MSG_LENGTH - Local, type byte - The length of the test message, in bytes, that is to be sent to the UNID.

Procedures

DET_DEST - Determine Destination - A procedure that determines the destination of the Datagram from the UNID.

LD_TAB_HSKP - Load Table Housekeep - Housekeeps a specified table after a new 128-byte data block has been loaded into the table.

LOAD_OUTGOING_DATA - A procedure that loads the remainder of the Datagram with outgoing data.

MAIN - This is the main procedure that drives all the other procedures through their proper sequencing.

ROUTE_IN - A procedure that routes in a 128-byte data block from the UNID and transmits it to the proper outgoing port.

ROUTE_OUT - A procedure that Routes out a 128-byte Datagram to the UNID.

SRVC_TAB_HSKP - Service Table Housekeep - Housekeeps a specified table whenever a 128-byte data block is removed from the table.

TRANSMIT_DATA - A procedure to send a 128-byte data block out an identified port.

MODULE ZILOG_LIB

The purpose of this assembly language module is to provide the Zilog MCZ 1/25 Computer System with the necessary library routines not currently available with PLZ/SYS.

Constants

None.

Variables

None.

Procedures

- MOVSEQ - Move Sequence - A procedure to move a block of data from one area in memory to another.
- RECSEQ - Receive Sequence - A procedure that receives a sequence of bytes from an identified port.
- SNDSEQ - Send Sequence - A procedure that sends a block of data out an identified port.
- TRNMIT - Transmit - A procedure that transmits the 128-byte data block to the local Host.

MODULE ZILOG_PHYSICAL

The purpose of this assembly language module is to support the Zilog Host at the Physical Level with the required input/output drivers.

Constants

None.

Variables

None.

Procedures

- INTUS0 - A procedure that initializes the 8251USART-0 located on the Zilog SIB board.
- INTUS1 - A procedure that initializes the 8251 USART-0 located on the Zilog SIB board.
- VECINT - A procedure to initialize the I/O vector interrupt process.
- ZRTR01 - A procedure that handles any receive interrupts from the UNID (J-112 of the Zilog).
- ZRTR02 - A procedure that handles any receive interrupts from the UNID (J-113 of the Zilog).
- ZRTRN - A procedure that handles any transmit interrupts to the UNID.

MODULE ZILOG_TABLES

The purpose of this PLZ language module is to provide the Zilog Operating System with the tables required for storing the 128-byte data blocks to/from the UNID.

Constants

DATA_SIZE - Number of bytes of data in a table.

DATA_TABLE_SIZE - Number of bytes in a table.

ERROR_TABLE_SIZE - Number of bytes in the error table.

PACKETS_IN_TABLE - Number of data packets in a data table.

Variables

ERROR_TABLE - Global, type array - A table to keep track of errors.

INCOMING_DATA_BLOCK - Global, type array - A table that contains an incoming Datagram for processing by the Zilog Host.

OUTGOING_DATA_BLOCK - Global, type array - A table that contains an outgoing Datagram the is being processed by the Zilog Host.

ZL01NE - Global, type integer - Pointer for the next available position within the table ZL01TB.

ZL01NS - Global, type integer - Pointer for the next byte to be serviced within the table ZL01TB.

ZL01SZ - Global, type integer - Size of the ZL01TB table.

ZL01TB - Global, type array - Local input table which interfaces with the UNID through Zilog port J112.

ZL02NE - Global, type integer - Pointer for the next available position within the table ZL02TB.

ZL02NS - Global, type integer - Pointer for the next byte to be serviced within the table ZL02TB.

ZL02SZ - Global, type integer - Size of the ZL02TB table.

ZL02TB - Global, type array - Local input table which interfaces with the UNID through Zilog port J113.

Procedures

INIT_ZILOG_TABLES - Initialize Zilog Tables. Sets up the Zilog tables and initializes the pointers to zeros.

MODULE ZILOG_TRANSPORT

The purpose of this PLZ module is to provide the Zilog Host with the required software to implement the Transport Layer. Currently, the Transport Layer just builds a dummy Datagram Header.

Constants

DATA_SIZE - Number of bytes of data in an input data table.

H19_COMMAND_PORT - Command port address for the H19 Terminal.

H19_DATA_PORT - Data port address for the H19 Terminal.

PRINTER_COMMAND_PORT - Command port address for the Spinwriter.

PRINTER_DATA_PORT - Data port address for the Spinwriter.

Variables

ACKNOWLEDGEMENT_NUMBER_FOUR - Global, type byte - Contains the last octet(8-bits) of the Datagram Header's Acknowledgement Field.

ACKNOWLEDGEMENT_NUMBER_ONE - Global, type byte - Contains the first octet(8-bits) of the Datagram Header's Acknowledgement Field.

ACKNOWLEDGEMENT_NUMBER_THREE - Global, type byte - Contains the third octet(8-bits) of the Datagram Header's Acknowledgement Field.

ACKNOWLEDGEMENT_NUMBER_TWO - Global, type byte - Contains the second octet(8-bits) of the Datagram Header's Acknowledgement Field.

C_FIELD_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's C-Field.

C_FIELD_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's C-Field.

CHECKSUM_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Checksum Field.

CHECKSUM_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Checksum Field.

CONTROL_BITS - Global, type byte - Contains the Datagram Header's Control Field information.

DATA_OFFSET - Global, type byte - Contains the Datagram Header's Data Offset Field value.

DESTINATION_PORT_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Destination Port Field.

DESTINATION_PORT_LOW - Global, type byte, - Contains the lower 8-bits of the Datagram Header's Destination Port Field.

FLAGS - Global, type byte - Contains the 3-bit Flag Field of the Datagram Header.

FRAGMENT_OFFSET_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Fragment Offset Field.

FRAGMENT_OFFSET_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Fragment Offset Field.

H_FIELD_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's H-Field.

H_FIELD_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's H-Field.

HEADER_CHECKSUM_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Header Checksum Field.

HEADER_CHECKSUM_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Header Checksum Field.

IDENTIFICATION_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Identification Field.

IDENTIFICATION_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Identification Field.

IHF_PADDING - Global, type byte - Contains the 8-bits of zeros used to pad the IHF portion of the Datagram Header.

IHL - Global, type byte - Contains the Datagram Internet Header Length.

OPTION - Global, type byte - Contains any option information used within the Datagram Header.

PROTOCOL - Global, type byte - Contains the protocol information of the Datagram.

RESERVED_HIGH - Global, type byte - Contains the first 4-bits of the Datagram Header's Reserved Field.

RESERVED_LOW - Global, type byte - Contains the last 2-bits of the Datagram Header's Reserved Field.

S_FIELD_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's S-Field.

S_FIELD_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's S-Field.

SECURITY_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Security Field.

SECURITY_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Security Field.

SEQUENCE_NUMBER_FOUR - Global, type byte - Contains the last octet(8-bits) of the Datagram Header's Sequence Number Field.

SEQUENCE_NUMBER_ONE - Global, type byte - Contains the first octet(8-bits) of the Datagram Header's Sequence Number Field.

SEQUENCE_NUMBER_THREE - Global, type byte - Contains the third octet(8-bits) of the Datagram Header's Sequence Number Field.

SEQUENCE_NUMBER_TWO - Global, type byte - Contains the second octet(8-bits) of the Datagram Header's Sequence Number Field.

SOURCE_PORT_HIGH - Global, type byte - Contains the upper 8-bits of the Datagram Header's Source Port Field.

SOURCE_PORT_LOW - Global, type byte - Contains the lower 8-bits of the Datagram Header's Source Port Field.

TCC_FIELD_ONE - Global, type byte - Contains the first octet(8-bits) of the Datagram Header's TCC-Field.

TCC_FIELD_THREE - Global, type byte - Contains the third
octet(8-bits) of the Datagram Header's
TCC-Field.

TCC_FIELD_TWO - Global, type byte - Contains the second
octet(8-bits) of the Datagram Header's
TCC-Field.

TCP_PADDING - Global, type byte - Contains the zeros
used to pad the TCP portion of the Datagram
Header.

TIME_TO_LIVE - Global, type byte - Contains the Datagram
Time to Live information.

TOTAL_LENGTH_HIGH - Global, type byte - Contains the
upper 8-bits of the Datagram Header's Total
Length Field.

TOTAL_LENGTH_LOW - Global, type byte - Contains the
lower 8-bits of the Datagram Header's Total
Length Field.

TYPE_OF_SERVICE - Global, type byte - Contains the
Datagram Type of Service information.

URGENT_POINTER_HIGH - Global, type byte - Contains the
upper 8-bits of the Datagram Header's Urgent
Pointer Field.

URGENT_POINTER_LOW - Global, type byte - Contains the
lower 8-bits of the Datagram Header's Urgent
Pointer Field.

VERSION - Global, type byte - Contains the version
number of the Datagram.

WINDOW_HIGH - Global, type byte - Contains the upper
8-bits of the Datagram Header's Window Field.

WINDOW_LOW - Global, type byte - Contains the lower
8-bits of the Datagram Header's Window Field.

Procedures

BUILD_TRANSPORT_HEADER - A procedure that builds a dummy
Datagram.

DATAGRAM_ACKNOWLEDGEMENT_NUMBER - A procedure that
inserts or extracts the Acknowledgement Number
from the Datagram Header.

DATAGRAM_C_FIELD - A procedure that inserts or extracts the C-Field information from the Datagram Header.

DATAGRAM_CHECKSUM - A procedure that inserts or extracts the Cchecksum information from the Datagram Header.

DATAGRAM_CONTROL_BITS - A procedure that inserts or extracts the Control information from the Datagram Header.

DATAGRAM_DATA_OFFSET - A procedure that inserts or extracts the Data Offset value from the Datagram Header.

DATAGRAM_DESTINATION_ADDRESS - A procedure that inserts or extracts the Destination Address from the Datagram Header.

DATAGRAM_DESTINATION_PORT - A procedure that inserts or extracts the Destination Port Address from the Datagram Header.

DATAGRAM_FLAGS - A procedure that inserts or extracts any flag information from the Datagram Header.

DATAGRAM_FRAGMENT_OFFSET - A procedure that inserts or extracts the Fragment Offset value from the Datagram Header.

DATAGRAM_H_FIELD - A procedure that inserts or extracts the H-Field information from the Datagram Header.

DATAGRAM_HEADER_CHECKSUM - A procedure that inserts or extracts the Header Checksum information from the Datagram Header.

DATAGRAM_IDENTIFICATION - A procedure that inserts or extracts the Identification information from the Datagram Header.

DATAGRAM_IHF_PADDING - A procedure that pads the rest of the IHF portion of the Datagram Header with zeros (to an even 4-byte boundary).

DATAGRAM_IHL - A procedure that inserts or extracts the Internet Header Length (IHL) from the Datagram Header.

DATAGRAM_OPTION - A procedure that inserts or extracts any Option information included in the Datagram Header.

DATAGRAM_PROTOCOL - A procedure that inserts or extracts the Protocol information from the Datagram Header.

DATAGRAM_RESERVED - A procedure that inserts or extracts the Reserved information from the Datagram Header.

DATAGRAM_S_FIELD - A procedure that inserts or extracts the S-Field information from the Datagram Header.

DATAGRAM_SECURITY - A procedure that inserts or extracts the Security information from the Datagram Header.

DATAGRAM_SEQUENCE_NUMBER - A procedure that inserts or extracts the Sequence Number of the Datagram.

DATAGRAM_SOURCE_ADDRESS - A procedure that inserts or extracts the Source Address from the Datagram Header.

DATAGRAM_SOURCE_PORT - A procedure that inserts or extracts the Source Port Address from the Datagram Header.

DATAGRAM_TCC_FIELD - A procedure that inserts or extracts the TCC-Field information from the Datagram Header.

DATAGRAM_TCP_PADDING - A procedure that pads the TCP portion of the Datagram Header with zeros (to an even 4-byte boundary).

DATAGRAM_TIME_TO_LIVE - A procedure that inserts or extracts the Datagram's Time to Live information.

DATAGRAM_TOTAL_LENGTH - A procedure that inserts or extracts the Datagram's Total Length.

DATAGRAM_TYPE_OF_SERVICE - A procedure that inserts or extracts the Datagram's Type of Service information.

DATAGRAM_URGENT_POINTER - A procedure that inserts or extracts the Urgent Pointer information from the Datagram Header.

DATAGRAM_VERSION - A procedure that inserts or extracts the Version of the Datagram.

DATAGRAM_WINDOW - A procedure that inserts or extracts the Window information from the Datagram Header.

EXTRACT_TRANSPORT_HEADER - A procedure that extracts the header information from an incoming Datagram.

4. Data Dictionary Cross Reference:

This section contains a cross reference of all the constants, variables, and procedures used in all modules of the UNID and ZILOG system software. The identifiers are arranged in alphabetical order and lists the specific modules discussed previously.

<u>Identifier</u>	<u>Modules</u>
ACKNOWLEDGE_A	N.MAIN_U1,N.MAIN_U2
ACKNOWLEDGE_B	N.MAIN_U1,N.MAIN_U.
ACKNOWLEDGEMENT_	
NUMBER_FOUR	ZILOG_TRANSPORT
ACKNOWLEDGEMENT_	
NUMBER_ONE	ZILOG_TRANSPORT
ACKNOWLEDGEMENT_	
NUMBER_THREE	ZILOG_TRANSPORT
ACKNOWLEDGEMENT_	
NUMBER_TWO	ZILOG_TRANSPORT
A_DATA	N.INSIO_U1,N.INSIO_U2
A_STAT	N.INSIO_U1,N.INSIO_U2
B_DATA	N.INSIO_U1,N.INSIO_U2
B_STAT	N.INSIO_U1,N.INSIO_U2
BITS	L.MAIN_U1,L.MAIN_U2
BUILD_I_FRAME	N.MAIN_U1,N.MAIN_U2
BUILD_I_PACKET	L.MAIN_U1,L.MAIN_U2
BUILD_S_FRAME	N.MAIN_U1,N.MAIN_U2
BUILD_TRANSPORT_	
HEADER	ZILOG_APPLICATION,ZILOG_TRANSPORT
C_FIELD_HIGH	ZILOG_TRANSPORT
C_FIELD_LOW	ZILOG_TRANSPORT
CHECKSUM_HIGH	ZILOG_TRANSPORT
CHECKSUM_LOW	ZILOG_TRANSPORT
CONCMD	L.MAIN_U1,L.MAIN_U2
CONCMD	N.MAIN_U1,N.MAIN_U2
CONCTC	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
CONDAT	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
CONTROL_BITS	ZILOG_TRANSPORT
CONTROL_CODE	L.MAIN_U1,L.MAIN_U2, ZILOG_APPLICATION
COUNTRY_CODE	L.MAIN_U1,L.MAIN_U2
CTC0	UNID_INIT_ONE,UNID_INIT_TWO
CTC_0	N.INSIO_U1,N.INSIO_U2
CTC1	UNID_INIT_ONE,UNID_INIT_TWO, UNID_ONE,UNID_TWO
CTC_1	N.INSIO_U1,N.INSIO_U2
CTC_2	N.INSIO_U1,N.INSIO_U2
CTC_3	N.INSIO_U1,N.INSIO_U2
CTCNOA	N.MAIN_U1,N.MAIN_U2
CTCNOB	N.MAIN_U1,N.MAIN_U2

<u>Identifier (cont)</u>	<u>Modules (cont)</u>
DATAGRAM_	
ACKNOWLEDGEMENT_	
NUMBER	ZILOG_TRANSPORT
DATAGRAM_C_FIELD	ZILOG_TRANSPORT
DATAGRAM_CHECKSUM	ZILOG_TRANSPORT
DATAGRAM_CONTROL_	
BITS	ZILOG_TRANSPORT
DATAGRAM_DATA_OFFSET	ZILOG_TRANSPORT
DATAGRAM_DESTINATION	
_PORT	ZILOG_TRANSPORT
DATAGRAM_FLAGS	ZILOG_TRANSPORT
DATAGRAM_FRAGMENT_	
OFFSET	ZILOG_TRANSPORT
DATAGRAM_H_FIELD	ZILOG_TRANSPORT
DATAGRAM_HEADER_	
CHECKSUM	ZILOG_TRANSPORT
DATAGRAM_	
IDENTIFICATION	ZILOG_TRANSPORT
DATAGRAM_IHF_PADDING	ZILOG_TRANSPORT
DATAGRAM_IHL	ZILOG_TRANSPORT
DATAGRAM_LENGTH	ZILOG_APPLICATION
DATAGRAM_OPTION	ZILOG_TRANSPORT
DATAGRAM_PROTOCOL	ZILOG_TRANSPORT
DATAGRAM_RESERVED	ZILOG_TRANSPORT
DATAGRAM_S_FIELD	ZILOG_TRANSPORT
DATAGRAM_SECURITY	ZILOG_TRANSPORT
DATAGRAM_SEQUENCE_	
NUMBER	ZILOG_TRANSPORT
DATAGRAM_SOURCE_PORT	ZILOG_TRANSPORT
DATAGRAM_TCC_FIELD	ZILOG_TRANSPORT
DATAGRAM_TCP_PADDING	ZILOG_TRANSPORT
DATAGRAM_TIME_TO_	
LIVE	ZILOG_TRANSPORT
DATAGRAM_TOTAL_	
LENGTH	ZILOG_TRANSPORT
DATAGRAM_TYPE_OF_	
SERVICE	ZILOG_TRANSPORT
DATAGRAM_URGENT_	
POINTER	ZILOG_TRANSPORT
DATAGRAM_VERSION	ZILOG_TRANSPORT
DATAGRAM_WINDOW	ZILOG_TRANSPORT
DATA_OFFSET	ZILOG_TRANSPORT
DATA_SIZE	L.MAIN_U1,L.TAB_U1,L.MAIN_U2, L.TAB_U2,ZILOG_APPLICATION, ZILOG_TABLES,ZILOG_TRANSPORT
DATA_TABLE	ZILOG_APPLICATION
DATA_TABLE_SIZE	L.MAIN_U1,L.MAIN_U2
DESTINATION	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
DESTINATION_ADDRESS	L.MAIN_U1,L.MAIN_U2
DESTINATION_	
ADDRESS_FOUR	ZILOG_APPLICATION,ZILOG_TRANSPORT

<u>Identifier (cont)</u>	<u>Modules (cont)</u>
DESTINATION_	
ADDRESS_ONE	ZILOG_APPLICATION, ZILOG_TRANSPORT
DESTINATION_	
ADDRESS_THREE	ZILOG_APPLICATION, ZILOG_TRANSPORT
DESTINATION_	
ADDRESS_TWO	ZILOG_APPLICATION, ZILOG_TRANSPORT
DESTINATION_PORT_	
HIGH	ZILOG_TRANSPORT
DESTINATION_PORT_	
LOW	ZILOG_TRANSPORT
DET_DEST	ZILOG_APPLICATION
DET_DEST_FOUR	L.MAIN_U1, L.MAIN_U2
DET_DEST_LL	L.MAIN_U1, L.MAIN_U2
DET_DEST_LN	L.MAIN_U1, N.MAIN_U1, L.MAIN_U2, N.MAIN_U2
DET_DEST_NN	N.MAIN_U1, N.MAIN_U2
DET_DEST_ONE	L.MAIN_U1, N.MAIN_U1, L.MAIN_U2, N.MAIN_U2
DET_DEST_THREE	L.MAIN_U1, L.MAIN_U2
DET_DEST_TWO	L.MAIN_U1, N.MAIN_U1, L.MAIN_U2, N.MAIN_U2
ERROR_TABLE	ZILOG_TABLES
ERROR_TABLE_SIZE	ZILOG_APPLICATION, ZILOG_TABLES
FALSE	N.MAIN_U1, N.MAIN_U2, N.INSIO_U1, N.INSIO_U2
FLAGS	ZILOG_TRANSPORT
FRAGMENT_OFFSET_	
HIGH	ZILOG_TRANSPORT
FRAGMENT_OFFSET_	
LOW	ZILOG_TRANSPORT
FRAMES_IN_TABLE	N.MAIN_U1, N.TAB_U1, N.MAIN_U2, N.TAB_U2
FRAME_SIZE	N.MAIN_U1, N.TAB_U1, U.SHTAB_U1, N.MAIN_U2, N.TAB_U2, U.SHTAB_U2
FRAME_TABLE_SIZE	N.MAIN_U1, N.TAB_U1, U.SHTAB_U1, N.MAIN_U2, N.TAB_U2, U.SHTAB_U2
H_FIELD_HIGH	ZILOG_TRANSPORT
H_FIELD_LOW	ZILOG_TRANSPORT
H19_COMMAND_PORT	ZILOG_APPLICATION, ZILOG_TRANSPORT
H19_DATA_PORT	ZILOG_APPLICATION, ZILOG_TRANSPORT
HEADER_CHECKSUM_HIGH	ZILOG_TRANSPORT
HEADER_CHECKSUM_LOW	ZILOG_TRANSPORT
HOST_CODE	L.MAIN_U1, L.MAIN_U2
IDENTIFICATION_HIGH	ZILOG_TRANSPORT
IDENTIFICATION_LOW	ZILOG_TRANSPORT
IHF_PADDING	ZILOG_TRANSPORT
IHL	ZILOG_TRANSPORT
INCOMING_DATA_BLOCK	ZILOG_TABLES, ZILOG_APPLICATION, ZILOG_TRANSPORT
INIT_L_TAB	L.TAB_U1, L.TAB_U2
INIT_N_TAB	N.TAB_U1, N.TAB_U2
INIT_ZILOG_TABLES	ZILOG_TABLES

<u>Identifier (cont)</u>	<u>Modules (cont)</u>
INIURT	L.VINT_U1,L.VINT_U2
INPUT_SEQ_BIT	N.MAIN_U1,N.MAIN_U2
INSIO	N.INSIO_U1,N.INSIO_U2
INTUS0	ZILOG_PHYSICAL
INTUS1	ZILOG_PHYSICAL
INTVEC	UNID_INIT_ONE,UNID_INIT_TWO
INVINT	L.VINT_U1,L.VINT_U2
IX	L.MAIN_U1,L.MAIN_U2,N.MAIN_U1, N.MAIN_U2,ZILOG_APPLICATION, ZILOG_TABLES
J112_DATA_PORT	ZILOG_APPLICATION
J113_DATA_PORT	ZILOG_APPLICATION
LCLCNE	L.TAB_U1,L.TAB_U2
LCLCNS	L.TAB_U1,L.TAB_U2
LCLCSZ	L.TAB_U1,L.TAB_U2
LCLCTB	L.TAB_U1,L.TAB_U2
LCNTNE	L.TAB_U1,U.SHTAB_U1,L.TAB_U2, U.SHTAB_U2
LCNTNS	L.TAB_U1,U.SHTAB_U1,L.TAB_U2, U.SHTAB_U2
LCNTSZ	L.TAB_U1,U.SHTAB_U1,L.TAB_U2, U.SHTAB_U2
LCNTTB	L.TAB_U1,U.SHTAB_U1,L.TAB_U2, U.SHTAB_U2
LC01NE	L.TAB_U1,L.TAB_U2
LC01NS	L.TAB_U1,L.TAB_U2
LC01SZ	L.TAB_U1,L.TAB_U2
LC01TB	L.TAB_U1,L.TAB_U2
LC02NE	L.TAB_U1,L.TAB_U2
LC02NS	L.TAB_U1,L.TAB_U2
LC02SZ	L.TAB_U1,L.TAB_U2
LC02TB	L.TAB_U1,L.TAB_U2
LC03NE	L.TAB_U1,L.TAB_U2
LC03NS	L.TAB_U1,L.TAB_U2
LC03SZ	L.TAB_U1,L.TAB_U2
LC03TB	L.TAB_U1,L.TAB_U2
LC04NE	L.TAB_U1,L.TAB_U2
LC04NS	L.TAB_U1,L.TAB_U2
LC04SZ	L.TAB_U1,L.TAB_U2
LC04TB	L.TAB_U1,L.TAB_U2
L_RI_DEST_ERR	L.MAIN_U1,L.MAIN_U2
L_RO_DEST_ERR	L.MAIN_U1,L.MAIN_U2
LD_TAB_HSKP	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION
LOAD_OUTGOING_DATA	ZILOG_APPLICATION
MAIN	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION
MAX_COUNTRY_CODE	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
MAX_NETWORK_CODE	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
MAXNOA	N.MAIN_U1,N.MAIN_U2

Identifier (cont)Modules (cont)

MAXNOB	N.MAIN_U1,N.MAIN_U2
MOVSEQ	U.LIB_U1,U.LIB_U2,ZILOG_LIB
NET_RI_DEST_ERR	N.MAIN_U1,N.MAIN_U2
NET_RO_DEST_ERR	N.MAIN_U1,N.MAIN_U2
NETWORK_CODE	L.MAIN_U1,L.MAIN_U2
NTLCNE	U.SHTAB_U1,U.SHTAB_U2
NTLCNS	U.SHTAB_U1,U.SHTAB_U2
NTLCSZ	U.SHTAB_U1,U.SHTAB_U2
NTLCTB	U.SHTAB_U1,U.SHTAB_U2
NT01NE	N.TAB_U1,N.TAB_U2
NT01NS	N.TAB_U1,N.TAB_U2
NT01SZ	N.TAB_U1,N.TAB_U2
NT01TB	N.TAB_U1,N.TAB_U2
NT02NE	N.TAB_U1,N.TAB_U2
NT02NS	N.TAB_U1,N.TAB_U2
NT02SZ	N.TAB_U1,N.TAB_U2
NT02TB	N.TAB_U1,N.TAB_U2
OPTION	ZILOG_TRANSPORT
OUTFRAME_CHA_NE	N.TAB_U1,N.TAB_U2
OUTFRAME_CHA_NS	N.TAB_U1,N.TAB_U2
OUTFRAME_CHA_SZ	N.TAB_U1,N.TAB_U2
OUTFRAME_CHA_TB	N.TAB_U1,N.TAB_U2
OUTFRAME_CHB_NE	N.TAB_U1,N.TAB_U2
OUTFRAME_CHB_NS	N.TAB_U1,N.TAB_U2
OUTFRAME_CHB_SZ	N.TAB_U1,N.TAB_U2
OUTFRAME_CHB_TB	N.TAB_U1,N.TAB_U2
OUTGOING_DATA_BLOCK	ZILOG_APPLICATION,ZILOG_TABLES, ZILOG_TRANSPORT
PACKET_SIZE	L.MAIN_U1,N.MAIN_U1,U.SHTAB_U1 L.MAIN_U2,N.MAIN_U2,U.SHTAB_U2
PACKETS_IN_TABLE	L.MAIN_U1,L.TAB_U1,N.MAIN_U1, U.SHTAB_U1,L.MAIN_U2,L.TAB_U2, N.MAIN_U2,U.SHTAB_U2,ZILOG_TABLES
PACKET_TABLE_SIZE	L.MAIN_U1,L.TAB_U1,U.SHTAB_U1 L.MAIN_U2,L.TAB_U2,U.SHTAB_U2
PICADD	L.VINT_U1,L.VINT_U2
PICCMD	L.VINT_U1,L.VINT_U2
PORT_ADDRESS	ZILOG_APPLICATION
PORT_CODE_HIGH	ZILOG_APPLICATION
PORT_CODE_LOW	ZILOG_APPLICATION
PRINTER_COMMAND_PORT	ZILOG_APPLICATION,ZILOG_TRANSPORT
PRINTER_DATA_PORT	ZILOG_APPLICATION,ZILOG_TRANSPORT
PROTOCOL	ZILOG_TRANSPORT
RCVRDY	UNID_ONE,UNID_TWO
RECCHA	N.INSIO_U1,N.INSIO_U2
RECCHB	N.INSIO_U1,N.INSIO_U2
RECSEQ	U.LIB_U1,U.LIB_U2,ZILOG_LIB
RESERVED_HIGH	ZILOG_TRANSPORT
RESERVED_LOW	ZILOG_TRANSPORT
ROUTE_IN	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION

Identifier (cont)Modules (cont)

ROUTE_OUT	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION
S_FIELD_HIGH	ZILOG_TRANSPORT
S_FIELD_LOW	ZILOG_TRANSPORT
SECURITY_HIGH	ZILOG_TRANSPORT
SECURITY_LOW	ZILOG_TRANSPORT
SEQ_BIT_A	N.MAIN_U1,N.MAIN_U2
SEQ_BIT_B	N.MAIN_U1,N.MAIN_U2
SEQUENCE_NUMBER_FOUR	ZILOG_TRANSPORT
SEQUENCE_NUMBER_ONE	ZILOG_TRANSPORT
SEQUENCE_NUMBER_THREE	ZILOG_TRANSPORT
SEQUENCE_NUMBER_TWO	ZILOG_TRANSPORT
SNDSEQ	U.LIB_U1,U.LIB_U2,ZILOG_LIB
SOURCE_ADDRESS	L.MAIN_U1,L.MAIN_U2
SOURCE_ADDRESS_FOUR	ZILOG_APPLICATION,ZILOG_TRANSPORT
SOURCE_ADDRESS_ONE	ZILOG_APPLICATION,ZILOG_TRANSPORT
SOURCE_ADDRESS_THREE	ZILOG_APPLICATION,ZILOG_TRANSPORT
SOURCE_ADDRESS_TWO	ZILOG_APPLICATION,ZILOG_TRANSPORT
SOURCE_PORT_HIGH	ZILOG_TRANSPORT
SOURCE_PORT_LOW	ZILOG_TRANSPORT
SRVC_TAB_HSKP	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION
STARTUP_HDR	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2,ZILOG_APPLICATION
STAT_NBR	L.MAIN_U1,N.MAIN_U2,U.SHTAB_U1, L.MAIN_U2,N.MAIN_U2,U.SHTAB_U2
STCTC2	N.INSIO_U1,N.INSIO_U2
STCTC3	N.INSIO_U1,N.INSIO_U2
TABLE	L.MAIN_U1,L.MAIN_U2,N.MAIN_U1, N.MAIN_U2,ZILOG_APPLICATION, ZILOG_TRANSPORT
TC1	N.INSIO_U1,N.INSIO_U2
TC2	N.INSIO_U1,N.INSIO_U2
TC3	N.INSIO_U1,N.INSIO_U2
TCC_FIELD_ONE	ZILOG_TRANSPORT
TCC_FIELD_THREE	ZILOG_TRANSPORT
TCC_FIELD_TWO	ZILOG_TRANSPORT
TCP_PADDING	ZILOG_TRANSPORT
TDAADD	L.MAIN_U1,L.MAIN_U2
TEST_MSG_LENGTH	ZILOG_APPLICATION
THIS_COUNTRY_CODE	L.MAIN_U1,L.MAIN_U2
THIS_SEQ_BIT	N.MAIN_U1,N.MAIN_U2
THIS_UNID_NBR	L.MAIN_U1,N.MAIN_U1,L.MAIN_U2, N.MAIN_U2
TIMCHA	N.MAIN_U1,N.MAIN_U2
TIMCHB	N.MAIN_U1,N.MAIN_U2
TIME_DELAY_CHA	N.MAIN_U1,N.MAIN_U2
TIME_DELAY_CHB	N.MAIN_U1,N.MAIN_U2
TIME_TO_LIVE	ZILOG_TRANSPORT
TOTAL_LENGTH_HIGH	ZILOG_TRANSPORT
TOTAL_LENGTH_LOW	ZILOG_TRANSPORT

<u>Identifier (cont)</u>	<u>Modules (cont)</u>
TPRADD	L.MAIN_U1,L.MAIN_U2
TRANSFER	UNID_INIT_ONE,UNID_INIT_TWO, UNID_ONE,UNID_TWO
TRANSMIT_DATA	ZILOG_APPLICATION
TRNMIT	L.VINT_U1,N.INSIO_U1,L.VINT_U2, N.INSIO_U2,ZILOG_LIB
TRNMIT_PKT	L.MAIN_U1,L.MAIN_U2
TRUE	N.MAIN_U1,N.MAIN_U2
TYPE_OF_SERVICE	ZILOG_TRANSPORT
UNIDIN	UNID_INIT_ONE,UNID_INIT_TWO
URGENT_POINTER_HIGH	ZILOG_TRANSPORT
URGENT_POINTER_LOW	ZILOG_TRANSPORT
URTR01	L.VINT_U1,L.VINT_U2
URTR02	L.VINT_U1,L.VINT_U2
URTR03	L.VINT_U1,L.VINT_U2
URTR04	L.VINT_U1,L.VINT_U2
USART	UNID_INIT_ONE,UNID_INIT_TWO, UNID_ONE,UNID_TWO
U01DAT	L.MAIN_U1,L.MAIN_U2
URTRN	L.VINT_U1,L.VINT_U2
U02DAT	L.MAIN_U1,L.MAIN_U2
U03DAT	L.MAIN_U1,L.MAIN_U2
U04DAT	L.MAIN_U1,L.MAIN_U2
VECADD	UNID_INIT_ONE,UNID_INIT_TWO
VECINT	ZILOG_PHYSICAL
VERSION	ZILOG_TRANSPORT
WINDOW_HIGH	ZILOG_TRANSPORT
WINDOW_LOW	ZILOG_TRANSPORT
XMITCA	N.INSIO_U1,N.INSIO_U2
XMITCB	N.INSIO_U1,N.INSIO_U2
ZL01NE	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL01NS	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL01SZ	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL01TB	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL02NE	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL02NS	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL02SZ	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZL02TB	ZILOG_TABLES,ZILOG_PHYSICAL, ZILOG_APPLICATION
ZRTR01	ZILOG_PHYSICAL
ZRTR02	ZILOG_PHYSICAL
ZRTTRN	ZILOG_PHYSICAL

Appendix Summary

This appendix summarizes all the constants, variables, and procedures used in the DELNET software for the UNID and the ZILOG MCZ 1/25 Computer System.

As a reminder, as the software changes within the UNID in respect to its location within the DELNET and the types of local Hosts it will service, the values given in this appendix will differ.

APPENDIX F

Software Listings

Introduction

This appendix contains five sections. The first section contains the software listings of the UNID start-up programs. The second section is the software listing of the shared data tables between the UNIDs local and network sides. The third section is the Local Operating System software listings of UNID#2 (UNID#1 is the same except for THIS_UNID_NBR and minor textual differences). The fourth section is the Network Operating System software listings of UNID#2 (UNID#1 is the same except for THIS_UNID_NBR and minor textual differences). The fifth and last section contains the DELNET Monitor (Zilog MCZ 1/25 Computer System) software.

Table of Contents

<u>Section</u>	<u>Page</u>
Section I: UNID Start-up Modules.....	F-2
UNID_INIT_TWO.....	F-4
UNID_TWO.....	F-9
Section II: UNID Shared Modules.....	F-26
U.LIB_U2.....	F-27
U.SHTAB_U2.....	F-35
Section III: UNID Local Operating System Modules..	F-39
L.MAIN_U2.....	F-40
L.TAB_U2.....	F-81
L.VINT_U2.....	F-85
Section IV: UNID Network Operating System Modules.	F-109
N.INSIO_U2.....	F-110
N.MAIN_U2.....	F-143
N.TAB_U2.....	F-178
Secton V: DELNET Monitor Operating System Modules.	F-181
ZILOG_APPLICATION.....	F-182
ZILOG_LIB.....	F-200
ZILOG_PHYSICAL.....	F-210
ZILOG_TABLES.....	F-224
ZILOG_TRANSPORT.....	F-228

Appendix F Section I

This section of Appendix F contains the software listings which comprise the programs used to start-up the UNIDs.

In order to enable the ZILOG MCZ 1/25 Computer System to be able to pass data to the UNIDs in a networking environment the setup programs developed by Baker (Ref 1) had to be modified.

Each UNID has to have its own port, enabling the ZILOG MCZ 1/25 Computer System to access both UNIDs without cable transfers. To do this, two setup programs were created using what Baker designed as a base. The first set is UNID_INIT_ONE and UNID_ONE while the second set is UNID_INIT_TWO and UNID_TWO. The primary difference between the two programs is which USART and CTC on the ZILOG MCZ 1/25 Computer System is used.

Following is the software listings (refer to Figure F-1 for the corresponding structure charts) with extensive documentation of UNID_INIT_TWO and UNID_TWO (with notes as to which has to be changed depending on which UNID is affected).

a. UNID_INIT_TWO (UNID_INIT_ONE): This program wakes up the USART connected to J-112 (J-113) and enables the desired UNID to communicate via interrupts. For each UNID in the DELNET there must be a host computer handling its network software therefore, UNID_INIT_TWO must be customized for each host. Each UNID_INIT_XXX program will be called UNID_INIT_unid number. Presently, there are two modules developed: UNID_INIT_ONE and UNID_INIT_TWO.

b. UNID_TWO (UNID_ONE): This program is used to transfer the Local and Network Operating System software from the ZILOG MCZ 1/25 Computer System to the desired UNID. As with UNID_INIT_TWO this software has to be customized depending on which UNID is desired. Each UNID_XXX program will be called UNID_unid number. Presently, there are two modules developed: UNID_ONE and UNID_TWO.

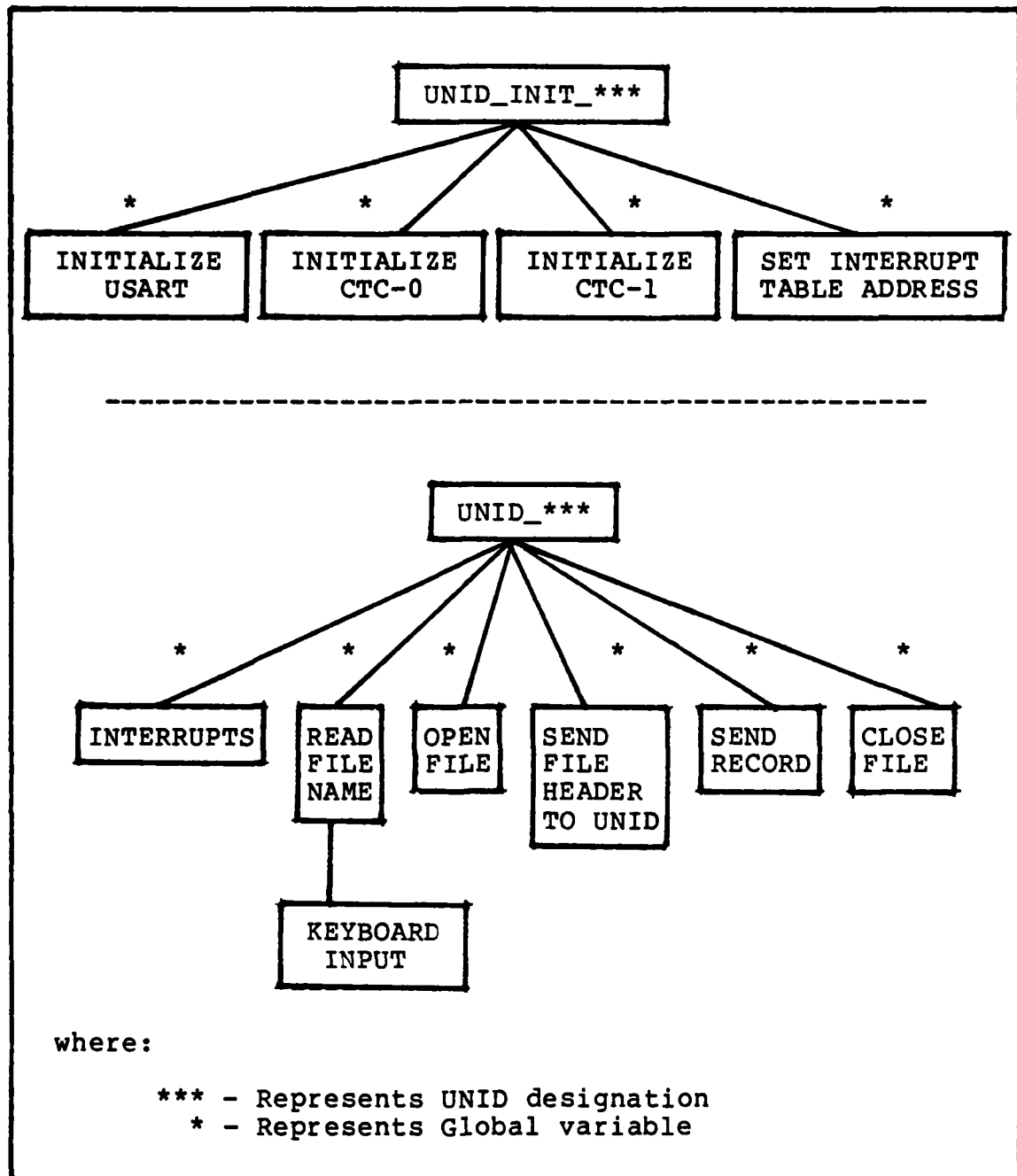


Figure F-1. UNID Initialization and File Transfer Procedures


```

;      This section is the initialization for the selected USART.

UNIDIN:      LD B,3
              LD A,0
              OUT (USART+1),A
THREE:      DJNZ THREE
              ;SEND OUT 3 ZEROS TO RESET USART:
              ;RESET SELECTED USART
              ;AND A RESET COMMAND
              ;SEND OUT MODE BYTE
              ;SEND OUT COMMAND BYTE
              IN A,(USART)
              ;CLEAR THE RECEIVER BUFFER

;      This section initializes CTC-0 Channel-0 on the selected USART.

              LD A,CONMOD
              OUT (CTC0),A
              LD A,RATE19
              OUT (CTC0),A
              ;SEND OUT COUNTER MODE TO:
              ;CHANNEL 0
              ;SEND OUT COUNTER VALUE TO:
              ;CHANNEL 0

;      This section initializes CTC-1 Channel-1 on the selected USART.

              LD A,INTVEC
              OUT (CTC1),A
              LD A,INTCW
              OUT (CTC1),A
              ;SET UP TO INTERRUPT WHEN THE RECEIVER
              ;ON THE SELECTED USART HAS A CHARACTER
              ;INTERRUPT ON EVERY CHARACTER
              LD A,1

```

```

OUT (CTC1),A      ;FOR THE SELECTED USART

```

```

; This section sets the address in the interrupt table.
;

```

```

LD HL,TRANSFER    ;GET THE MAIN PROGRAM START ADDRESS
LD (VECADD),HL    ;PUT ADDRESS INTO INTERRUPT TABLE
RET

```

```

; Equates

```

```

USART EQU 8CH
CTC0 EQU 80H
CTC1 EQU 84H

```

```

RESET EQU 01000000B
MODE EQU 01001110B

```

```

;USART-0 PORT ADDRESS
;CHANNEL-0 CTC-0 PORT ADDRESS
;CHANNEL-0 CTC-1 PORT ADDRESS

```

```

;RESET FOR USART
;MODE BYTE

```

```

; BIT 0 & 1 - 00 SYNC MODE
;           - 01 ASYNC MODE, 1X
;           BAUD RATE FACTOR
;           - 10 ASYNC MODE, 16X
;           BAUD RATE FACTOR
;           - 11 ASYNC MODE, 64X
;           BAUD RATE FACTOR
;           BIT 2 & 3 - 00 5 BITS/CHARACTER
;           - 01 6 BITS/CHARACTER
;           - 10 7 BITS/CHARACTER
;           - 11 8 BITS/CHARACTER
;           BIT 4 - 0 PARITY DISABLED
;           - 1 PARITY ENABLED
;           BIT 5 - 0 ODD PARITY
;           - 1 EVEN PARITY
;           BIT 6 & 7 - 00 INVALID
;           - 01 1 STOP BIT
;           - 10 1 1/2 STOP BITS
;

```

```

;          - 11 2 STOP BITS

;COMMAND BYTE
;  BIT 0 - 1 ENABLE TRANSMISSION
;          0 DISABLE TRANSMISSION
;  BIT 1 - 1 DTR OUTPUT FORCE TO 0
;  BIT 2 - 1 ENABLE RECEIVER
;          0 DISABLE RECEIVER
;  BIT 3 - 1 FORCE TxD LOW
;          0 NORMAL OPERATION
;  BIT 4 - 1 RESET ALL ERROR FLAGS
;  BIT 5 - 1 RTS OUTPUT FORCE TO 0
;  BIT 6 - 1 RESET FORMAT
;  BIT 7 - 1 ENTER HUNT MODE

;CHANNEL CONTROL WORD
;  BIT 0 - HAS TO BE 1
;  BIT 1 - 1 RESET CHANNEL
;          0 CONTINUE OPERATION
;  BIT 2 - 1 TIME CONSTANT NEXT
;          0 NO TIME CONSTANT
;  BIT 3 - SEE SIB BOOK
;  BIT 4 - 1 POSITIVE EDGE
;          0 NEGATIVE EDGE
;  BIT 5 - 1 PRESCALER FACTOR 256
;          0 PRESCALER FACTOR 16
;  BIT 6 - 1 COUNTER MODE SELECT
;          0 TIMER MODE SELECT
;  BIT 7 - 1 ENABLE INTERRUPTS
;          0 DISABLE INTERRUPTS

;TIME CONSTANT

;CHANNEL CONTROL WORD
;INTERRUPT PAGE ADDRESS

```

;SEE NOTE CONCERNING INTERRUPT ADDRESS

TRANSFER: EQU 0F000H
VECADD: EQU 1340H

;MAIN PROGRAM STARTING ADDRESS
;INTERRUPT TABLE ADDRESS

;SEE NOTE CONCERNING INTERRUPT ADDRESS

; ----- END OF PROGRAM -----

```

*****P 53*****
;*****DATE ORIGINATED: 1 JUNE 81*****
;*****DATE LAST MODIFIED:*****
;PROLOGUE      MODULE UNID_TWO
;
;THIS MODULE IS USED TO TRANSFER THE LOCAL AND NETWORK
;OPERATING SYSTEM SOFTWARE OF THE UNID FROM THE ZILOG MCZ 1/25 COMPUTER
;SYSTEM TO THE DESIRED UNID.
;CURRENTLY, THERE ARE TWO SIMILAR MODULES. THIS ONE IS UNID_TWO.S
;TO TRANSFER THE SOFTWARE TO UNID 2 WHILE THE OTHER IS UNID_ONE.S TO
;TRANSFER THE SOFTWARE TO UNID 1.
;
;NOTE: 1. CURRENTLY, EACH UNID MUST HAVE A RESIDENT HOST THAT KEEPS THE
;UNID's SOFTWARE. FOR INITIAL START-UP, THE USART OF THE HOST
;MUST CONTAIN THE PORT ADDRESS TO WHICH UNID CHANNEL-1 IS CONNECTED
;INTO. FOR THE ZILOG MCZ 1/25 COMPUTER SYSTEM THE USART ADDRESSES
;AVAILABLE FOR UNID CONNECTION ARE:
;    A. USART-0   8CH   UNID 2 CONNECTED
;    B. USART-1   8EH   UNID 1 CONNECTED
;
;2. FOR THE ZILOG MCZ 1/25 COMPUTER SYSTEM THE FOLLOWING CTC'S ARE
;PROGRAMMED FOR INITIAL SOFTWARE SYSTEM TRANSFER:
;    A. USART-0   CTC-1 CHANNEL-0   84H   UNID 2 CONNECTED
;    B. USART-1   CTC-1 CHANNEL-1   85H   UNID 1 CONNECTED
;
;*****
;*H MAIN PROGRAM
TRANSFER:
LD (SAVSTK),SP
LD SP,STACK
CALL INIT
CALL RDNAME
CALL OPENFL
CALL TEST
CALL SNHEAD
LD DE,(RECCNT)

;SAVE SYSTEM STACK POINTER
;INITIALIZE STACK POINTER
;TURN OFF SIB AND ON SYSTEM INTERRUPTS
;READ FILE NAME FROM SERIAL PORT
;OPEN THE FILE
;MAKE SURE IT IS A PROCEDURE FILE
;SEND THE FILE HEADER
;GET THE NUMBER OF RECORDS TO BE SENT

```

NEXT?

```
CALL SKIPFD
CALL READRC
LD BC,ONEREC
LD HL,BUFFER
CALL SEND
LD HL,(RECCNT)
DEC HL
LD (RECCNT),HL
LD A,H
OR L
JR NZ,NEXT?
CALL CLOSE
JP RETURN
```

*H READ NAME

RDNAME:

```
CALL GETFIRST
```

WAIT:

```
LD HL,NAMELT
IN A,(USART+1)
BIT RCVRDY,A
JR Z,WAIT
```

RDBYTE:

```
LD HL,NAMEBF
```

```
LD B,A
IN A,(USART+1)
BIT RCVRDY,A
JR Z,RDBYTE
```

```
IN A,(USART)
```

```
;SKIP FORWARD ONE RECORD
;READ ONE RECORD
;SET BUFFER LENGTH
;LOAD ADDRESS OF BUFFER
;SEND THE RECORD
;GET THE PRESENT RECORD COUNT
;DECREMENT RECORD COUNTER
;SAVE RECORD COUNT
;IS THIS THE LAST RECORD TO BE SENT?
; NO, THEN GET ANOTHER RECORD
; YES, THEN CLOSE THE FILE
```

```
;CHECK FOR PROPER LEADING CHARACTER (FF)
```

```
;GET THE ADDRESS OF THE NAME LENGTH
```

```
;IS THERE SOMETHING IN THE RECEIVER?
```

```
; NO, THEN WAIT
; YES, THEN GET IT
;GET THE FILE NAME LENGTH
;AND SAVE IT
```

```
;GET THE ADDRESS OF THE NAME BUFFER
```

```
;LOAD NAME LENGTH COUNTER
;READ THE NAME INTO THE NAME BUFFER
;IS THE NEXT CHARACTER IN THE RECEIVER?
; NO, THEN WAIT
```

```
; YES, THEN GET THE CHARACTER
```

```

LD (HL),A      ; PUT THE CHARACTER INTO NAME BUFFER
INC HL         ; INCREMENT NAME BUFFER POINTER
DJNZ RDBYTE    ; IS THERE ANOTHER CHARACTER?
               ; YES, THEN GET IT
RET            ; NO, THEN RETURN

GETFIRST:      ;GET ONE BYTE
               ;IS IT THE PROPER LEADING CHARACTER (FF)
               ; YES, THEN RETURN
               ; NO, THEN GET OUT OF PROGRAM
JP RETURN

```

```

; EQUATES

USART          EQU 8CH
RCVRDY         EQU 1

*H OPEN FILE

OPENFL:        ;LOAD OPEN REQUEST VECTOR ADDRESS
               ;INTO VECTOR
               ;LOAD NUMBER OF ATTRIBUTES TO BE MOVED

               ;OPEN FILE
               ;FLAG FILE AS OPEN

```



```

;
; BIT 6 - ERASE PROTECTION
; BIT 5 - LOCKED
; BIT 4 - SECRET
; BIT 3 - RANDOM
; BIT 2 - FORCE
; BIT 1 - RESERVED
; BIT 0 - RESERVED
;
; START ADDRESS
; RESERVED
; DATE OF CREATION
; DATE LAST WRITTEN
; PROGRAM SEGMENT INFORMATION
; UNUSED
; LOWEST SEGMENT ADDRESS
; HIGHEST SEGMENT ADDRESS
; STACK SIZE

```

```

STADDR:      DEFS 2
              DEFS 2
              DEFS 8
              DEFS 8
              DEFS 64D
              DEFS 18D
              DEFS 2
              DEFS 2
              DEFS 2
              DEFS 2

```

```

SEGMENT:
LOWADD:
HIGHAD:
STKSIZ:

```

```

; EQUATES

```

```

OPRQCD:      EQU 4
FILELU:      EQU 10
OPENUD:      EQU 0
FILEOP:      EQU 1

```

```

*H TEST FILE

```

```

TEST:
LD A, (FILEAT)
BIT 7,A
RET NZ

```

```

LD HL,WRGFIL
LD (MESSAGE),HL
LD HL,LWRGFL
LD (LMESS),HL

```

```

; OPEN REQUEST
; FILE LOGICAL UNIT
; OPEN FOR UPDATE
; FILE OPEN FLAG

; GET FILE TYPE
; IS IT A PROCEDURE FILE?
; YES, THEN RETURN

; NO, THEN SEND OUT A ERROR MESSAGE

```

```

LD HL, CONOUT
LD (VECTOR), HL
CALL SYSCAL

CALL CLOSE
;CLOSE THE FILE

POP HL
;ADJUST THE STACK

JP RETURN

; MESSAGE
WRGFIL:  DEFM 'FILE NOT OF TYPE - PROCEDURE'
DEFB 0DH
LWRGFL:  EQU $-WRGFIL

*H SEND HEADER
SNHEAD:  LD DE, BUFFER
;GET THE ADDRESS OF SEND BUFFER

LD A, (RECCNT)
LD (DE), A
;GET RECORD COUNT AND
INC DE
;PUT IT IN BUFFER
LD A, (RECCNT+1)
LD (DE), A
INC DE

LD A, (RECLNT)
LD (DE), A
;GET RECORD LENGTH AND
INC DE
;PUT IT IN BUFFER
LD A, (RECLNT+1)
LD (DE), A
INC DE

LD A, (STADDR)
;GET START ADDRESS AND

```

```

LD (DE),A
INC DE
LD A,(STADDR+1)
LD (DE),A
;PUT IT IN BUFFER

LD HL,SEGMENT
INC DE
LD BC,64D
LDIR
;BEGINNING OF SEGMENT ADDRESSES

LD BC,HEADLN
LD HL,BUFFER
;LENGTH OF SEGMENT INFORMATION STORAGE
;PUT IN BUFFER

CALL SEND
;SET UP FOR SEND

CALL SEND
;SEND HEADER

RET

; EQUATES

HEADLN EQU 70D
;HEADER LENGTH

*H READ ONE RECORD

READRC:
LD HL,ONEREC
LD (RDCTRV+DATA1T),HL
;GET RECORD LENGTH

LD HL,RDCTRV
LD (VECTOR),HL
;GET READ REQUEST VECTOR ADDRESS
;INTO VECTOR

CALL SYSCAL
;READ ONE RECORD

RET

```

```

;          STORAGE

RDCTRV      DEFB FILELU
             DEFB READCT
             DEFW BUFFER
             DEFW 0
             DEFW 0
             DEFW 0
             DEFB 0
             DEFW 0

;          LOGICAL UNIT
;          ;REQUEST CODE
;          ;DATA TRANSFER AREA
;          ;DATA LENGTH
;          ;COMPLETION RETURN ADDRESS
;          ;ERROR RETURN ADDRESS
;          ;COMPLETION CODE
;          ;SUPPLEMENTAL PARAMETER

;          EQUATES

ONEREC      EQU 128
READCT      EQU 1EH
DATALT      EQU 4

*H SEND BUFFER

SEND:
             IN A,(USART+1)
             BIT TRNRDY,A
             JR Z,SEND

             LD A,(HL)
             OUT (USART),A

             DEC BC
             INC HL

             LD A,B
             OR C
             JR NZ,SEND

             RET

;          ;IS SELECTED USART READY TO TRANSMIT
;          ; NO, THEN WAIT
;          ; YES, SEND ONE BYTE

             ;DECREMENT LENGTH COUNTER
             ;INCREMENT BUFFER POINTER

             ;WAS THIS THE LAST CHARACTER IN BUFFER?
             ; NO, THEN SEND ANOTHER CHARACTER
             ; YES, THEN RETURN

```

```

;      STORAGE
BUFFER      DEFS ONEREC
;      EQUATES
TRNRDY      EQU 0
;H SKIP FORWARD ONE RECORD
SKIPFD:      LD HL,SKIPRV
              LD (VECTOR),HL
              LD HL,MOVONE
              LD (NUMREC),HL
              CALL SYSCAL
              RET
;      STORAGE
SKIPRV      DEFB FILELU
              DEFB SKPFWD
              DEFW 0
              DEFW 0
              DEFW 0
              DEFW 0
              DEFB 0
              DEFW 0
;      EQUATES
;OUTPUT BUFFER
;TRANSMITTER READY BIT
;LOAD SKIP FORWARD REQUEST VECTOR ADD
;INTO VECTOR
;LOAD NUMBER OF RECORDS TO SKIP
;SKIP ONE RECORD FORWARD
;LOGICAL UNIT
;REQUEST CODE
;DATA TRANSFER AREA
;DATA LENGTH (NO. OF RECORDS TO MOVE)
;COMPLETION RETURN ADDRESS
;ERROR RETURN ADDRESS
;COMPLETION CODE
;SUPPLEMENTAL PARAMETER

```

SKPFWD	EQU 24H	;SKIP FORWARD REQUEST CODE
MOVONE	EQU 01H	;MOVE ONE RECORD
*H SYSTEM CALL		
SYSICAL:	LD IY, (VECTOR)	;GET ADDRESS OF VECTOR
	CALL SYSTEM	;CALL SYSTEM WITH REQUEST
	LD A, (IY+COMPCD)	;GET COMPLETION CODE
	BIT ERRBIT, A	;WAS THERE AN ERROR?
	CALL NZ, ERROR	; YES, THEN GO TO ERROR ROUTINE
	RET	; NO, THEN RETURN
ERROR:	LD HL, ERRCOD	;CONVERT ERROR CODE TO ASCII AND SAVE
	CALL CONTOA	
	LD A, (IY+REQCOD)	;GET REQUEST CODE
	LD HL, PRSTRQ	;CONVERT TO ASCII AND SAVE
	CALL CONTOA	
	LD HL, ERRMSG	;SET UP TO SEND MESSAGE
	LD (MESSAGE), HL	
	LD HL, LERRMG	
	LD (LMESS), HL	
	LD HL, CONOUT	;LOAD CONSOLE OUT REQUEST VECTOR
	LD (VECTOR), HL	
	CALL SYSICAL	;SEND OUT ERROR MESSAGE
	LD A, (FILEST)	;IS THE FILE OPEN?
	CP FILEOP	

CALL Z,CLOSE	; YES, THEN CLOSE THE FILE
JP RETURN	; NO, THEN RETURN FROM INTERRUPT
CONTOA:	PUSH AF
	RRA
	RRA
	RRA
	RRA
	;GET THE HIGH ORDER BITS
	;SAVE NUMBER
	CALL CON4BT
	;CONVERT HIGH ORDER BITS
	POP AF
	;GET NUMBER
	INC HL
	;INCREMENT STORAGE POINTER
	CALL CON4BT
	;CONVERT LOWER ORDER BITS
	RET
CON4BT:	AND 0FH
	;MASK OFF LOW ORDER BITS
	CP 10
	JR C,YES
	ADD A,7
	; NO
YES:	ADD A,30H
	LD (HL),A
	;STORE CONVERTED NUMBER
	RET


```

;          STORAGE

CONOUT      DEFB CONSOT
MESSAGE      DEFB WRTLIN
LMESS        DEFW 0
              DEFW 0
              DEFW 0
              DEFW RIO
              DEFB 0
              DEFW 0

VECTOR       DEFW 0

```

```

;LOGICAL UNIT
;REQUEST CODE
;DATA TRANSFER AREA
;DATA LENGTH
;COMPLETION RETURN ADDRESS
;ERROR RETURN ADDRESS
;COMPLETION CODE
;SUPPLEMENTAL PARAMETER
;REQUEST PARAMETER VECTOR

```

```

;          EQUATES

ERRBIT      EQU 6
COMPCD      EQU 10
SYSTEM      EQU 1403H
REQCOD      EQU 1
CONSOT      EQU 2
WRTLIN      EQU 10H
RIO         EQU 1400H

;          MESSAGE

ERRMSG:      DEFM 'ERROR '
ERRCOD      DEFW 0
PRSTRQ      DEFM ' DURING REQUEST CODE '
LERRMG      DEFW 0
              DEFB 0DH
              EQU $-ERRMSG

```

```

;COMPLETION CODE ERROR BIT
;COMPLETION CODE OFFSET
;SYSTEM ENTRY POINT
;REQUEST CODE OFFSET
;CONSOLE OUT LOGICAL UNIT
;WRITE LINE REQUEST CODE
;SYSTEM REENTRY POINT

```

```

*H INITIALIZATION AND RETURN
RETURN:      LD A,INTON
              OUT (CTCL),A
              CALL RESREG
              LD SP,(SAVSTK)
              RETI
;            EQUATES
INTON:       EQU 11010001B
;
CLOSE:       LD HL,CLSRV
              LD (VECTOR),HL
              CALL SYSCAL
              LD A,FILECL
              LD (FILEST),A
              RET
;            STORAGE
CLSRV        DEFB FILELU
              DEFB CLSFIL
              DEFW 0
              DEFW 0
              DEFW 0
              DEFW 0
              DEFW 0
              ;LOGICAL UNIT
              ;REQUEST CODE
              ;DATA TRANSFER AREA
              ;DATA LENGTH
              ;COMPLETION RETURN ADDRESS
              ;ERROR RETURN ADDRESS
              ;TURN ON SIB INTERRUPTS
              ;RESTORE REGISTERS
              ;RESTORE SYSTEM STACK POINTER
              ;RETURN FROM INTERRUPT
              ;CHANNEL COMMAND WORD
              ;LOAD CLOSE REQUEST VECTOR ADDRESS
              ;INTO VECTOR
              ;CLOSE FILE
              ;FLAG FILE CLOSED

```

```

DEFB 0      ;COMPLETION CODE
DEFW 0      ;SUPPLEMENTAL PARAMETER

; EQUATES

CLSFIL      EQU 06      ;CLOSE FILE REQUEST CODE
FILECL      EQU 0      ;FILE CLOSED FLAG

INIT:        CALL SAVREG ;SAVE ALL REGISTERS

LD A,INTOFF ;TURN OFF SIB INTERRUPTS
OUT (CTCL),A

EI           ;ENABLE SYSTEM INTERRUPTS
RET

; EQUATES

INTOFF:      EQU 01010001B ;CHANNEL COMMAND WORD
CTCL:        EQU 84H       ;CTC-1 CH-0 PORT ADDRESS

;H REGISTER SAVE AND RESTORE

SAVREG:      LD (STKPTR),SP ;SAVE STACK POINTER

LD SP,SAVSTK ;SET SP TO BOTTOM OF SAVE STACK AREA

PUSH AF      ;SAVE MAIN REGISTERS
PUSH BC
PUSH DE
PUSH HL
PUSH IX

```

PUSH IY	
LD A,I	;SAVE I REGISTER
PUSH AF	
EX AF,AF'	;GET ALTERNATE REGISTERS
EXX	
PUSH AF	;SAVE ALTERNATE REGISTERS
PUSH BC	
PUSH DE	
PUSH HL	
EX AF,AF'	;GET MANY REGISTERS
EXX	
LD SP,(STKPTR)	;RESTORE STACK POINTER
RET	
RESREG:	
LD (STKPTR),SP	;SAVE STACK POINTER
LD SP,TOPSTK	;SET SP TO TOP OF SAVE REGISTER AREA
EX AF,AF'	;GET ALTERNATE REGISTER SET
EXX	
POP HL	;RESTORE ALTERNATE REGISTERS
POP DE	
POP BC	
POP AF	

```

EX AF,AF'
EXX

POP AF
LD I,A

POP IY
POP IX
POP HL
POP DE
POP BC
POP AF

LD SP, (STKPTR)

RET

```

```

;GET MAIN REGISTER SET

;RESTORE I REGISTER

;RESTORE MAIN REGISTERS

;RESTORE STACK POINTER

;ALTERNATE REGISTERS
; L'
; H'
; D' AND E'
; B' AND C'
; ACCUMULATOR' AND FLAGS'
;MAIN REGISTERS
; I AND FLAGS
; IY
; IX
; H AND L
; D AND E
; B AND C
; ACCUMULATOR AND FLAGS

```

```

; STORAGE FOR SAVING REGISTERS

```

```

TOPSTK:
DEFB 0
DEFB 0
DEFW 0
DEFW 0
DEFW 0

DEFW 0
DEFW 0
DEFW 0
DEFW 0
DEFW 0
DEFW 0
DEFW 0

```

SAVSTK:	DEFB 0	;SYSTEM STACK
	DEFB 0	; LOW BYTE
		; HIGH BYTE
STKPTR:	DEFS 2	;TEMPORARY STORAGE FOR STACK POINTER
STACK:	EQU \$+128	;ADDRESS OF TOP OF STACK
END:	END	;END OF PROGRAM
		;----- END OF PROGRAM -----

Appendix F Section II

This section of Appendix F contains the software listings for the shared data tables used by both the Local and Network side of the UNIDs.

```
*****  
; ***** DATE ORIGINATED: 26 OCT 82  
; *****  
; ***** DATE LAST MODIFIED:  
; *****  
; PROLOGUE - MODULE U.LIB_U2  
;  
;  
; THIS MODULE IS AN ASSEMBLY LIBRARY PACKAGE BUILT  
; TO SUPPORT PLZ/SYS SOFTWARE THAT WAS DEVELOPED TO OPERATE  
; OUTSIDE OF THE NORMAL ZILOG SUPPORTED ENVIRONMENT. ITS  
; PURPOSE IS TO PROVIDE THE NECESSARY INTERFACE AND LIBRARY  
; FUNCTIONS NOT DIRECTLY AVAILABLE WITH PLZ/SYS. THE MODULE  
; CURRENTLY CONSISTS OF PROCEDURES MOVSEQ, RECSEQ, AND SNDSEQ.  
;
```

GLOBAL MOVSEQ RECSEQ SNDSEQ

```

*****
*EJECT
*****
;PROCEDURE      MOVSEQ      MOVE A SEQUENCE OF BYTES IN MEMORY
;
;
;      THE PURPOSE OF THIS PROCEDURE IS TO MOVE A SEQUENCE
;      OF BYTES FROM ONE LOCATION IN MEMORY TO ANOTHER.
;
;      INPUT -      THIS PROCEDURE EXPECTS THREE VALUES: THE SOURCE (FROM)
;                  MEMORY ADDRESS, THE DESTINATION (TO) MEMORY ADDRESS, AND
;                  THE NUMBER OF BYTES TO BE TRANSFERED.  ALL THREE PARAMETERS
;                  ARE OF WORD LENGTH.
;
;      PROCESSING -  THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;                  FOR NORMALIZATION AT THE RETURN.  THE PUSH ESTABLISHES
;                  THE BASE LOCATION FOR THE STACK.  ALL INPUT PARAMETERS
;                  WILL BE OFFSET FROM THIS BASE LOCATION.
;                  THE NEXT SECTION RETRIEVES THE THREE INPUT
;                  PARAMETERS AND LOADS THEM INTO THE BC, DE, AND HL REG SETS.
;                  THE LDIR COMMAND WILL MOVE THE CONTENTS OF THE
;                  LOCATION INDICATED BY HL TO THE LOCATION INDICATED BY DE.
*****

```


IT WILL INCREMENT HL AND DE, AND DECREMENT BC. THIS
PROCEDURE WILL CONTINUE UNTIL BC IS EQUAL TO 0.

AFTER COMPLETION OF THE LDIR, IX IS RESTORED, THE
RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
MODULE IS EXECUTED.

OUTPUT - NONE.

INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
PLZ USER GUIDE, SECTION 7 FOR DETAILS.

NOTES - 1. THE NUMBER OF BYTES THAT CAN BE TRANSFERRED
BY THIS PROCEDURE IS LIMITED BY THE REGISTER SET SIZE. ATTEMPTING
TO TRANSFER A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
IN 16 BITS WILL HAVE UNPREDICTABLE RESULTS.

MOVSEQ: ; PROCEDURE TO MOVE A SEQUENCE OF BYTES
; STORE IX FOR RETURN

PUSH IX

LD IX,0 ; SET IX TO BASE OF STACK

ADD IX,SP

LD C,(IX+4)

LD B,(IX+5)

; LD BYTES TO MOVE

LD E,(IX+6)

LD D,(IX+7)

; LD DESTINATION ADDRESS

```

LD L,(IX+8)      ; LD SOURCE ADDRESS
LD H,(IX+9)

LDIR             ; MOVE BYTES

POP IX           ; RESTORE IX
POP HL           ; RECOVER RETURN ADDRESS

POP DE           ; DEALLOCATE STACK
POP DE
POP DE

JP (HL)          ; RETURN

```

```

*****
*EJECT
*****
;PROCEDURE      RECSEQ      RECEIVE SEQUENCE OF BYTES
;
;
;      THE PURPOSE OF THIS PROCEDURE IS TO RECEIVE A
;      SEQUENCE OF BYTES FROM AN IDENTIFIED PORT.
;
;INPUT -      THIS PROCEDURE EXPECTS FOUR INPUT VALUES: THE
;      USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;      THE STARTING MEMORY LOCATION OF WHERE TO SEND THE DATA,
;      AND THE NUMBER OF BYTES TO RECEIVE. THE TWO USART PORT
;      ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;      THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.
;
;PROCESSING - THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;      FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;      THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;      WILL BE OFFSET FROM THIS BASE LOCATION.
;      THE NEXT SECTION RETRIEVES THE FOUR INPUT

```



```

LD L, (IX+6)
LD H, (IX+7)
; LD ADDRESS FOR RECEIPT OF DATA

LD D, (IX+8)
; LD DATA PORT ADDRESS (CONDAT)
; SINCE USING BITS ONLY THE LOWER
; ORDER BITS ARE USED IN THE
; STATEMENTS ABOVE.

LD C, (IX+10)
; LD COMAND PORT ADDRESS (CONCMD)

RECLP1
IN A, (C)
BIT 1, A
JR Z, RECLP1
; WAIT UNTIL READY TO RECEIVE

LD C, D
LD A, (HL)
IN A, (C)
; LD DATA PORT ADDRESS
; LD DATA ADDRESS
; RECEIVE BYTE

LD (HL), A
INC HL
LD C, (IX+10)
DJNZ RECLP1
; STORE DATA
; ADVANCE DATA ADDRESS POINTER
; LD COMMAND PORT ADDRESS
; IF NMBR BYTES LEFT > 0, RECEIVE ANOTHER
; ELSE CONTINUE
POP IX
POP HL
; RESTORE IX
; RECOVER RETURN ADDRESS

POP DE
POP DE
POP DE
POP DE
; DEALLOCATE STACK

JP (HL)
; RETURN

```

```

;*****
;*****

```

```

*EJECT
;*****
;PROCEDURE      SNDSEQ      SEND SEQUENCE OF BYTES
;
;      THE PURPOSE OF THIS PROCEDURE IS TO SEND A
;      SEQUENCE OF BYTES TO AN IDENTIFIED PORT.
;
;INPUT -      THIS PROCEDURE EXPECTS FOUR INPUT VALUES: THE
;      USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;      THE STARTING MEMORY LOCATION OF THE DATA TO BE SENT,
;      AND THE NUMBER OF BYTES TO RECEIVE. THE TWO USART PORT
;      ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;      THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.
;
;PROCESSING - THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;      FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;      THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;      WILL BE OFFSET FROM THIS BASE LOCATION.
;      THE NEXT SECTION RETRIEVES THE FOUR INPUT
;      PARAMETERS AND LOADS THEM INTO THE B, C, D, AND HL REGS.
;      A SYNC LOOP IS NEXT FOR CHECKING READY STATUS. THIS
;      LOOP SYNCHRONIZES THE CODE WITH THE CHOSEN BAUD RATE. THE
;      ACTUAL OUTPUT CODE FOLLOWS WITH THE APPROPRIATE INCREMENT
;      AND DECREMENT OF POINTERS. IF THERE ARE BYTES REMAINING
;      TO BE SENT, A LOOP BACK TO THE SYNC LOOP CONTINUES THE
;      OUTPUT PROCESS. OTHERWISE, THE IX REG IS RESTORED, THE
;      RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
;      STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
;      MODULE IS EXECUTED.
;
;OUTPUT -      NONE.
;
;INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;      COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
;      PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND

```

AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
 OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
 PLZ USER GUIDE, SECTION 7 FOR DETAILS.

NOTES - 1. THE NUMBER OF BYTES THAT CAN BE SENT
 BY THIS PROCEDURE IS LIMITED BY THE REGISTER SIZE. ATTEMPTING
 TO SEND A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
 IN 8 BITS WILL HAVE UNPREDICTABLE RESULTS.

SNDSEQ: ; PROCEDURE TO SEND SEQUENCE OF BYTES

PUSH IX ; STORE IX FOR RETURN

LD IX,0 ; SET IX TO BASE OF STACK
 ADD IX,SP

LD B,(IX+4) ; LD BYTES TO SEND

LD L,(IX+6) ; LD ADDRESS OF DATA TO SEND
 LD H,(IX+7)

LD D,(IX+8) ; LD DATA PORT ADDRESS (CONDAT)

LD C,(IX+10) ; LD COMAND PORT ADDRESS (CONCMD)

IN A,(C) ; WAIT UNTIL READY TO TRANSMIT
 BIT 0,A
 JR Z,SNDLPL1

LD C,D ; LD DATA PORT ADDRESS
 LD A,(HL) ; LD DATA ADDRESS
 OUT (C),A ; SEND BYTE

INC HL ; ADVANCE DATA ADDRESS POINTER

SNDLPL1

```

LD C,(IX+10)
DJNZ SNDLPI
; LD COMMAND PORT ADDRESS
; IF NMBR BYTES LEFT > 0, SEND ANOTHER
; ELSE CONTINUE
POP IX
POP HL
; RESTORE IX
; RECOVER RETURN ADDRESS
; DEALLOCATE STACK

POP DE
POP DE
POP DE
POP DE

JP (HL)
; RETURN

```

```

*****
;*****
;*****

```

```

*****
MODULE U.SHTAB_U2  UNID SHARED TABLE MODULE      DATE ORIGINATED: 26 OCT 82
                                                    DATE LAST MODIFIED:

```

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE LOCAL OPERATING SYSTEM (L.OS) AND THE NETWORK OPERATING SYSTEM (N.OS) WITH THE TABLES SHARED FOR INTERFACE BETWEEN THE TWO PROCESSES. ADDITIONALLY, IT PROVIDES A STATUS TABLE FOR STATUS MONITORING. THIS PROCEDURE CONSISTS PRIMARILY OF TABLE DEFINITIONS WITH PROCESSING LIMITED TO THE INITIALIZATION OF THE DEFINED TABLES VIA INIT_U_SHTAB.

NOTES - 1. STATTB ENTRIES ARE AS FOLLOWS:

- 00 - CUMULATIVE LOC ROUTE_IN DEST ERRORS
- 01 - CUMULATIVE LOC ROUTE_OUT DEST ERRORS
- 02 - LOCAL CONTROL_CODE ERROR
- 03 - LOCAL COUNTRY_CODE ERROR
- 04 - LOCAL NETWORK_CODE ERROR
- 05 - LOCAL HOST_CODE ERROR
- 06 - LOCAL PORT_CODE ERROR
- 07 - NOT USED
- 08 - NOT USED
- 09 - NOT USED
- 10 - CUMULATIVE NET ROUTE_IN DEST ERRORS
- 11 - CUMULATIVE NET ROUTE_OUT DEST ERRORS
- 12 - NETWORK CONTROL_CODE ERROR
- 13 - NETWORK COUNTRY_CODE ERROR
- 14 - NETWORK NETWORK_CODE ERROR
- 15 - NETWORK HOST_CODE ERROR
- 16 - NETWORK PORT_CODE ERROR
- 17 - NOT USED
- 18 - NOT USED
- 19 - NOT USED

```

*****

```


U_SHTAB_U2 MODULE

CONSTANT

```
DATA_SIZE := 128 ! DATA FROM HOST IS IN 128-BYTE BLOCKS !
PACKET_SIZE := DATA_SIZE + 5 ! 5-BYTES OF HEADER !
PACKETS_IN_TABLE := 10
STAT_NBR := 20
PACKET_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
```

GLOBAL

```
LCNTTB ARRAY [PACKET_TABLE_SIZE BYTE]
LCNTNS INTEGER
LCNTNE INTEGER
LCNTSZ INTEGER
```

```
NTLCTB ARRAY [PACKET_TABLE_SIZE BYTE]
NTLCNS INTEGER
NTLCNE INTEGER
NTLCNZ INTEGER
```

```
STATTB ARRAY [STAT_NBR BYTE]
```

```
!*****
PROCEDURE INIT_U_SHTAB !*****
PROCEDURE TO INITIALIZE DATA TABLES
```

THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE
DATA TABLES SHARED BOTH BY THE LOCAL OPERATING
SYSTEM AND THE NETWORK OPERATING SYSTEM.

```
INPUT - NONE.
```

PROCESSING - THE PROCEDURE INITIALIZES THE LOCAL-TO-NETWORK
AND THE NETWORK-TO-LOCAL TABLES BY SETTING
THE NEXT-BYTE-TO-BE-SERVED AND THE NEXT-EMPTY-
BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO

A MULTIPLE OF PACKET_SIZE. THE PROCEDURE FINISHES BY CLEARING THE STATUS TABLE OF ALL STATUS INFORMATION.

OUTPUT - THE TABLE POINTERS AND STATUS TABLE AS NOTED UNDER PROCESSING ARE MODIFIED.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-L.MAIN_U2 IN THE LOCAL OPERATING SYSTEM.

NOTES - 1. ALL TABLES ARE 133-BYTES IN LENGTH (PACKET_SIZE).

INIT_U_SHTAB PROCEDURE

LOCAL

IX WORD

ENTRY

! INITIALIZE MEMORY TABLE INFORMATION

xxxxns - NEXT BYTE TO BE SERVICED

xxxxne - NEXT EMPTY BYTE

xxxxsz - SIZE OF TABLE

!

! INITIALIZE LOCAL-TO-NETWORK TABLE

!

LCNTNS := 0

LCNTNE := 0

LCNTSZ := PACKET_TABLE_SIZE

! INITIALIZE NETWORK-TO-LOCAL TABLE

!

NTLCNS := 0

NTLCNE := 0

NTLCNZ := PACKET_TABLE_SIZE

IX := 0 ! INITIALIZE STATUS TABLE TO ZEROS !

```

DO
  STATTB[IX] := 0
  IX += 1
  IF IX = STAT_NBR THEN EXIT FI
OD

  END INIT_U_SHTAB

!*****!
END U_SHTAB_U2
!*****!
!*****!

```

Appendix F Section III

This section of Appendix F contains the software listings which comprise the Local Operating System of UNID#2 (UNID#1 is the same except for THIS_UNID_NBR and minor textual differences).


```

PACKET_SIZE := DATA_SIZE + 5 ! DATA_PACKET = 5-BYTES HEADER !
PACKETS_IN_TABLE := 10
STAT_NBR := 20
! STATUS ENTRIES IN STATTB !
DATA_TABLE_SIZE := DATA_SIZE * PACKETS_IN_TABLE
PACKET_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE

```

! FOLLOWING ARE NETWORK DEFINED VARIABLES !

- ! NOTES: 1. THIS_UNID_NBR MUST REFLECT WHICH UNID THIS IS.
 2. THIS_COUNTRY_CODE MUST REFLECT THE AREA TO WHICH THIS UNID IS LOCATED.
 3. MAX_COUNTRY_CODE WILL INDICATE WHICH COUNTRY CODES ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR THE DELNET MONITOR.
 4. MAX_NETWORK_CODE WILL INDICATE HOW MANY UNIDS ARE CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
 5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO PHISTER'S THESIS APPENDIX C. !

```

THIS_UNID_NBR := 2 ! UNIQUE ADDRESS OF THIS UNID !
THIS_COUNTRY_CODE := 01 ! CC WHERE THIS UNID RESIDES !
MAX_COUNTRY_CODE := 01 ! INDICATES COUNTRY_CODES IN USE !
MAX_NETWORK_CODE := 3 ! INDICATES UNIDS OPERATIONAL IN NET !

```

```

EXTERNAL ! IN L.VINT_U2 !

```

```

INVINT PROCEDURE
TRNMIT PROCEDURE

```

```

EXTERNAL ! IN U.LIB_U2 !

```

```

MOVSEQ PROCEDURE (SRCADD PBYTE, DTDADD PBYTE, NUMBYT BYTE)
SNDSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)
RECSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)

```

```

EXTERNAL ! IN L.TAB_U2 !

```

INIT_L_TAB PROCEDURE

LC01TB ARRAY [DATA_TABLE_SIZE BYTE]
LC01NS INTEGER
LC01NE INTEGER
LC01SZ INTEGER

LC02TB ARRAY [DATA_TABLE_SIZE BYTE]
LC02NS INTEGER
LC02NE INTEGER
LC02SZ INTEGER

LC03TB ARRAY [DATA_TABLE_SIZE BYTE]
LC03NS INTEGER
LC03NE INTEGER
LC03SZ INTEGER

LC04TB ARRAY [DATA_TABLE_SIZE BYTE]
LC04NS INTEGER
LC04NE INTEGER
LC04SZ INTEGER

LCLCTB ARRAY [DATA_TABLE_SIZE BYTE]
LCLCNS INTEGER
LCLCNE INTEGER
LCLCSZ INTEGER

EXTERNAL ! IN U.SHTAB_U2 !

INIT_U_SHTAB PROCEDURE

LCNNTB ARRAY [PACKET_TABLE_SIZE BYTE]
LCNTNS INTEGER
LCNTNE INTEGER
LCNTSZ INTEGER

```

NTLCTB ARRAY [PACKET_TABLE_SIZE BYTE]
NTLCNS INTEGER
NTLCNE INTEGER
NTLCNZ INTEGER

STATTB ARRAY [STAT_NBR BYTE]

GLOBAL      ! GLOBAL VARIABLES !
TDAADD PBYTE      ! LOC CHANNEL TRANSMIT DATA ADDRESS !
TPRADD PBYTE      ! LOC CHANNEL TRANSMIT PORT ADDRESS !

INTERNAL      ! INTERNAL VARIABLES USED IN THIS MODULE !
DESTINATION WORD      ! DESTINATION OF THE PACKET !

STARTUP_HDR ARRAY [* BYTE] := '%R%L'
      'UNID 2 LOCAL OS%R%L'
      'VERSION 27 SEP 83%R%L'
      'EXECUTING%R%L'

! THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID !
TP_1 ARRAY[*BYTE] := '%R%LTP_1: ENTERING INIT_L_TAB PROCEDURE'
TP_2 ARRAY[*BYTE] := '%R%LTP_2: ENTERING INIT_U_SHTAB PROCEDURE'
TP_3 ARRAY[*BYTE] := '%R%LTP_3: ENTERING INVINT PROCEDURE'
TP_4 ARRAY[*BYTE] := '%R%LTP_4: STARTING ROUTE_IN-ROUTE_OUT LOOP'
TP_5 ARRAY[*BYTE] := '%R%LTP_5: ENTERING ROUTE_IN PROCEDURE'
TP_6 ARRAY[*BYTE] := '%R%LTP_6: DATA LOCATED IN LOCAL CHANNEL-1'
TP_7 ARRAY[*BYTE] := '%R%LTP_7: DATA IS LC01TB-TO-LCLCTB TRANSFER'
TP_8 ARRAY[*BYTE] := '%R%LTP_8: DATA IS LC01TB-TO-LCNTTB TRANSFER'
TP_9 ARRAY[*BYTE] := '%R%LTP_9: ERROR OCCURRED IN LOCAL CHANNEL-1 IN-PROCESSING'
TP_10 ARRAY[*BYTE] := '%R%LTP_10: DATA LOCATED IN LOCAL CHANNEL-2'
TP_11 ARRAY[*BYTE] := '%R%LTP_11: DATA IS LC02TB-TO-LCLCTB TRANSFER'
TP_12 ARRAY[*BYTE] := '%R%LTP_12: DATA IS LC02TB-TO-LCNTTB TRANSFER'
TP_13 ARRAY[*BYTE] := '%R%LTP_13: ERROR OCCURRED IN LOCAL CHANNEL-2 IN-PROCESSING'
TP_14 ARRAY[*BYTE] := '%R%LTP_14: DATA LOCATED IN LOCAL CHANNEL-3'

```



```

TP_15 ARRAY[*BYTE] := '%R%LTP_15: DATA IS LC03TB-TO-LCLCTB TRANSFER'
TP_16 ARRAY[*BYTE] := '%R%LTP_16: DATA IS LC03TB-TO-LCNTTB TRANSFER'
TP_17 ARRAY[*BYTE] := '%R%LTP_17: ERROR OCCURRED IN LOCAL CHANNEL-3 IN-PROCESSING'
TP_18 ARRAY[*BYTE] := '%R%LTP_18: DATA LOCATED IN LOCAL CHANNEL-4'
TP_19 ARRAY[*BYTE] := '%R%LTP_19: DATA IS LC04TB-TO-LCLCTB TRANSFER'
TP_20 ARRAY[*BYTE] := '%R%LTP_20: DATA IS LC04TB-TO-LCNTTB TRANSFER'
TP_21 ARRAY[*BYTE] := '%R%LTP_21: ERROR OCCURRED IN LOCAL CHANNEL-4 IN-PROCESSING'
TP_22 ARRAY[*BYTE] := '%R%LTP_22: ENTERING ROUTE_OUT PROCEDURE'
TP_23 ARRAY[*BYTE] := '%R%LTP_23: OUTGOING DATA IS IN LCLCTB'
TP_24 ARRAY[*BYTE] := '%R%LTP_24: DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-1'
TP_25 ARRAY[*BYTE] := '%R%LTP_25: DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-2'
TP_26 ARRAY[*BYTE] := '%R%LTP_26: DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-3'
TP_27 ARRAY[*BYTE] := '%R%LTP_27: DATA IN LCLCTB DESTINED FOR LOCAL CHANNEL-4'
TP_28 ARRAY[*BYTE] := '%R%LTP_28: ERROR OCCURRED IN LCLCTB OUT-PROCESSING'
TP_29 ARRAY[*BYTE] := '%R%LTP_29: OUTGOING DATA IS IN NTLCTB'
TP_30 ARRAY[*BYTE] := '%R%LTP_30: DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-1'
TP_31 ARRAY[*BYTE] := '%R%LTP_31: DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-2'
TP_32 ARRAY[*BYTE] := '%R%LTP_32: DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-3'
TP_33 ARRAY[*BYTE] := '%R%LTP_33: DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-4'
TP_34 ARRAY[*BYTE] := '%R%LTP_34: ERROR OCCURRED IN NTLCTB OUT-PROCESSING'
TP_35 ARRAY[*BYTE] := '%R%LTP_35: END OF ROUTE_IN-ROUTE_OUT LOOP'
TP_36 ARRAY[*BYTE] := '%R%LTP_36: HAVE EXITED ROUTE_IN-ROUTE_OUT LOOP'

```

INTERNAL

!*****
 PROCEDURE DET_DEST_ONE DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
 COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-1.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
 LOCATION OF THE PACKET TO BE EVALUATED.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMING DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE DESTINATION_ADDRESS CAN BE DETERMINED.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('LN' OR 'LL') ALONG WITH THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

- NOTES: 1. BYTE AND %0F WILL MASK OUT THE UPPER 4-BITS.
 2. BYTE AND %F0 WILL MASK OUT THE LOWER 4-BITS.
 3. BYTE / %01 WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
 4. BYTE / %10 WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

*****!

INTERNAL

DET_DEST_ONE PROCEDURE (TABLE WORD)
 RETURNS (DESTINATION WORD, DESTINATION_ADDRESS BYTE)

LOCAL

 BITS INTEGER
 CONTROL_CODE BYTE
 COUNTRY_CODE BYTE
 NETWORK_CODE BYTE
 HOST_CODE INTEGER

ENTRY

HOST_CODE := 0

```

CONTROL_CODE := LC01TB [LC01NS+16] AND %F0
IF CONTROL_CODE = 00
THEN
  COUNTRY_CODE := LC01TB [LC01NS+16] AND %0F
  IF COUNTRY_CODE <= MAX_COUNTRY_CODE
  THEN
    NETWORK_CODE := LC01TB [LC01NS+17] AND %F0
    IF (NETWORK_CODE / %10) <= MAX_NETWORK_CODE
    THEN
      IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR IF
        ((NETWORK_CODE / %10) <> THIS_UNID_NBR)
      THEN
        DESTINATION := 'LN'
        BITS := INTEGER (LC01TB[LC01NS+17] AND %0F) / %01
        HOST_CODE := BITS * 16
        BITS := INTEGER (LC01TB[LC01NS+18] AND %F0) / %10
        HOST_CODE := HOST_CODE + BITS
        IF (HOST_CODE >= 0) ANDIF (HOST_CODE <= 63) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %01 FI
          IF (HOST_CODE >= 64) ANDIF (HOST_CODE <= 127) THEN
            DESTINATION_ADDRESS := NETWORK_CODE OR %02 FI
          IF (HOST_CODE >= 128) ANDIF (HOST_CODE <= 191) THEN
            DESTINATION_ADDRESS := NETWORK_CODE OR %03 FI
          IF (HOST_CODE >= 192) ANDIF (HOST_CODE <= 255) THEN
            DESTINATION_ADDRESS := NETWORK_CODE OR %04 FI
          ELSE DESTINATION := 'LL'
        FI
      ELSE
        DESTINATION := 'ER'
        STATTB[04] += 1 ! INCREMENT NETWORK_CODE ERROR !
        STATTB[00] += 1 ! INCREMENT LOCAL ERROR COUNT !
      FI
    ELSE
      DESTINATION := 'ER'
      STATTB[03] += 1 ! INCREMENT COUNTRY_CODE ERROR !
    FI
  FI

```

```

        STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI
    ELSE
        ! IT IS AT THIS POINT THAT OTHER CONTROL CODES
        ! WILL BE INCORPORATED INTO THE NETWORK LAYER. !
        DESTINATION := 'ER'
        STATTB [02] += 1 ! INCREMENT CONTROL_CODE ERROR !
        STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI

```

END DET_DEST_ONE

!*****!

!*****!
 PROCEDURE DET_DEST_TWO DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
 COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-2.

INPUT - THE INPUT IS A TWO ASCII VALUE INDICATING THE TABLE LOCATION OF THE
 DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMING
 DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE
 AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S
 DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE
 UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
 DESTINATION_ADDRESS CAN BE DETERMINED.

OUTPUT- THIS PROCEDURE RETURNS THE DESTINATION ('LN' OR 'LL') ALONG WITH
 THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS
 IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

NOTES:

1. BYTE AND %0F WILL MASK OUT THE UPPER 4-BITS.	
2. BYTE AND %0 WILL MASK OUT THE LOWER 4-BITS.	
3. BYTE / %01 WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.	
4. BYTE / %10 WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.	

INTERNAL

LOCAL

ENTRY

```

DESTINATION := 'LN'
BITS := INTEGER (LC02TB[LC02NS+17] AND %0F)/%01
HOST_CODE := BITS * 16
BITS := INTEGER (LC02TB[LC02NS+18] AND %F0)/%10
HOST_CODE := HOST_CODE + BITS
IF (HOST_CODE>=0) .ANDIF (HOST_CODE<=63) THEN
  DESTINATION_ADDRESS := NETWORK_CODE OR %01 FI
  IF (HOST_CODE>=64) .ANDIF (HOST_CODE<=127) THEN
    DESTINATION_ADDRESS := NETWORK_CODE OR %02 FI
  IF (HOST_CODE>=128) .ANDIF (HOST_CODE<=191) THEN
    DESTINATION_ADDRESS := NETWORK_CODE OR %03 FI
  IF (HOST_CODE>=192) .ANDIF (HOST_CODE<=255) THEN
    DESTINATION_ADDRESS := NETWORK_CODE OR %04 FI
  ELSE DESTINATION := 'LL'
  FI
ELSE
  DESTINATION := 'ER'
  STATTB [04] += 1 ! INCREMENT NETWORK_CODE ERROR !
  STATTB [00] += 1 ! INCREMENT LOCAL CODE ERROR !
  FI
ELSE
  DESTINATION := 'ER'
  STATTB [03] += 1 ! INCREMENT COUNTRY_CODE ERROR !
  STATTB [00] += 1 ! INCREMENT LOCAL CODE ERROR !
  FI
ELSE
  ! IT IS AT THIS POINT THAT OTHER CONTROL CODES
  ! WILL BE INCORPORATED INTO THE NETWORK LAYER. !
  DESTINATION := 'ER'
  STATTB [02] += 1 ! INCREMENT CONTROL_CODE ERROR !
  STATTB [00] += 1 ! INCREMENT LOCAL CODE ERROR !
  FI

```

END DET_DEST_TWO

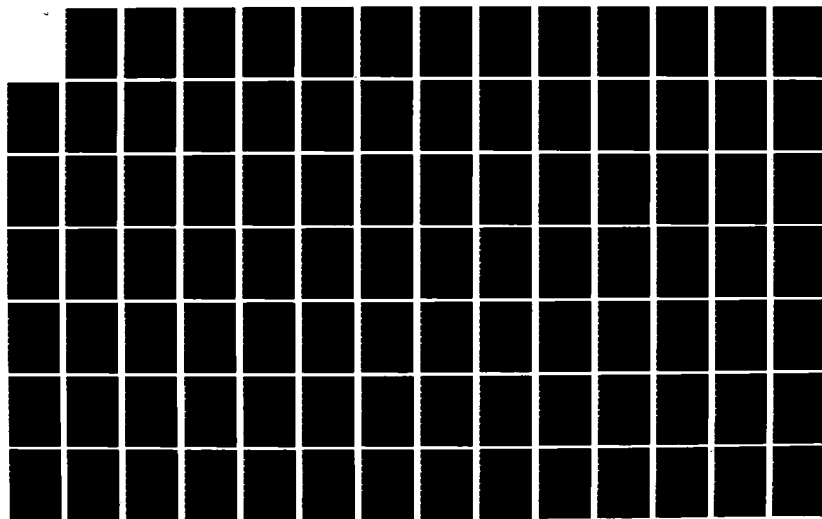
HD-A138 119

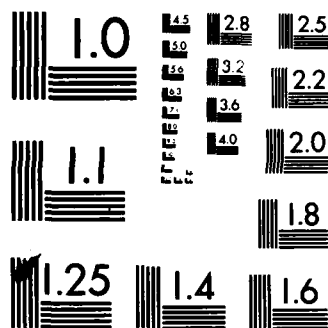
PROTOCOL STANDARDS AND IMPLEMENTATION WITHIN THE
DIGITAL ENGINEERING LABO... (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... P W PHISTER
DEC 83 AFIT/GE/EE/83D-58 F/G 17/2

2/4

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

!*****!
!*****!

PROCEDURE DET_DEST_THREE DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-3.

INPUT - THE INPUT IS A TWO ASCII VALUE INDICATING THE TABLE LOCATION OF
THE DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE
INCOMING DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE
COUNTRY_CODE AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE
DATA'S DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE
OF THE UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE
DESTINATION_ADDRESS CAN BE DETERMINED.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('LN' OR 'LL') ALONG WITH
THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS IS 21,
WHICH INDICATES UNID=2 AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

- NOTES: 1. BYTE AND %0F WILL MASK OUT THE UPPER 4-BITS.
2. BYTE AND %F0 WILL MASK OUT THE LOWER 4-BITS.
3. BYTE / %01 WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
4. BYTE / %10 WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

*****!
INTERNAL

DET_DEST_THREE PROCEDURE (TABLE WORD)
RETURNS (DESTINATION WORD, DESTINATION_ADDRESS BYTE)

LOCAL

 BITS INTEGER
CONTROL_CODE BYTE
COUNTRY_CODE BYTE
NETWORK_CODE BYTE
 HOST_CODE INTEGER

ENTRY

```
HOST_CODE := 0
CONTROL_CODE := LC03TB [LC03NS+16] AND %F0
IF CONTROL_CODE = 00
THEN
  COUNTRY_CODE := LC03TB [LC03NS+16] AND %0F
  IF COUNTRY_CODE <= MAX_COUNTRY_CODE
  THEN
    NETWORK_CODE := LC03TB [LC03NS+17] AND %F0
    IF (NETWORK_CODE / %10) <= MAX_NETWORK_CODE
    THEN
      IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR IF
        ((NETWORK_CODE / %10) <> THIS_UNID_NBR)
      THEN
        DESTINATION := 'LN'
        BITS := INTEGER (LC03TB [LC03NS+17] AND %0F) / %01
        HOST_CODE := BITS * 16
        BITS := INTEGER (LC03TB [LC03NS+18] AND %F0) / %10
        HOST_CODE := HOST_CODE + BITS
        IF (HOST_CODE >= 0) AND IF (HOST_CODE <= 63) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %01 FI
          IF (HOST_CODE >= 64) AND IF (HOST_CODE <= 127) THEN
            DESTINATION_ADDRESS := NETWORK_CODE OR %02 FI
          IF (HOST_CODE >= 128) AND IF (HOST_CODE <= 191) THEN
            DESTINATION_ADDRESS := NETWORK_CODE OR %03 FI
          IF (HOST_CODE >= 192) AND IF (HOST_CODE <= 255) THEN
```

```

        DESTINATION_ADDRESS := NETWORK_CODE OR $04 FI
        ELSE DESTINATION := 'LL'
        FI
    ELSE
        DESTINATION := 'ER'
        STATTB [04] += 1 ! INCREMENT NETWORK_CODE ERROR !
        STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI
    ELSE
        DESTINATION := 'ER'
        STATTB [03] += 1 ! INCREMENT COUNTRY_CODE ERROR !
        STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI
    ELSE
        ! IT IS AT THIS POINT THAT OTHER CONTROL CODES
        ! WILL BE INCORPORATED INTO THE NETWORK LAYER. !
        DESTINATION := 'ER'
        STATTB [02] += 1 ! INCREMENT CONTROL_CODE ERROR !
        STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI

```

END DET_DEST_THREE

```

!*****!
!*****!
PROCEDURE DET_DEST_FOUR  DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-4.

INPUT - THE INPUT IS A TWO ASCII VVALUE INDICATING THE TABLE LOCATION OF
THE DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMING

DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY_CODE AND NETWORK_CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK SIDE OF THE UNID THEN THE HOST_CODE IS ALSO EXTRACTED SO THAT THE DESTINATION_ADDRESS CAN BE DETERMINED.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('LN' OR 'LL') ALONG WITH THE DESTINATION_ADDRESS. AN EXAMPLE OF A DESTINATION_ADDRESS IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

- NOTES: 1. BYTE AND %0F WILL MASK OUT THE UPPER 4-BITS.
 2. BYTE AND %F0 WILL MASK OUT THE LOWER 4-BITS.
 3. BYTE / %01 WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
 4. BYTE / %10 WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

*****!

INTERNAL

DET_DEST_FOUR PROCEDURE (TABLE WORD)
 RETURNS (DESTINATION WORD, DESTINATION_ADDRESS BYTE)

LOCAL

BITS INTEGER
 CONTROL_CODE BYTE
 COUNTRY_CODE BYTE
 NETWORK_CODE BYTE
 HOST_CODE INTEGER

ENTRY

HOST_CODE := 0
 CONTROL_CODE := LC04TB [LC04NS+16] AND %F0

```

IF CONTROL_CODE = 00
THEN
  COUNTRY_CODE := LC04TB [LC04NS+16] AND %0F
  IF COUNTRY_CODE <= MAX_COUNTRY_CODE
  THEN
    NETWORK_CODE := LC04TB [LC04NS+17] AND %F0
    IF (NETWORK_CODE / %10) <= MAX_NETWORK_CODE
    THEN
      IF (COUNTRY_CODE <> THIS_COUNTRY_CODE) OR IF
        ((NETWORK_CODE / %10) <> THIS_UNID_NBR)
      THEN
        DESTINATION := 'LN'
        BITS := INTEGER (LC04TB[LC04NS+17] AND %0F)/%01
        HOST_CODE := BITS * 16
        BITS := INTEGER (LC04TB[LC04NS+18] AND %F0)/%10
        HOST_CODE := HOST_CODE + BITS
        IF (HOST_CODE>=0) ANDIF (HOST_CODE<=63) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %01 FI
        IF (HOST_CODE>=64) ANDIF (HOST_CODE<=127) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %02 FI
        IF (HOST_CODE>=128) ANDIF (HOST_CODE<=191) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %03 FI
        IF (HOST_CODE>=192) ANDIF (HOST_CODE<=255) THEN
          DESTINATION_ADDRESS := NETWORK_CODE OR %04 FI
        ELSE DESTINATION := 'LL'
      FI
    ELSE
      DESTINATION := 'ER'
      STATTB [04] += 1 ! INCREMENT NETWORK_CODE ERROR !
      STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI
  ELSE
    DESTINATION := 'ER'
    STATTB [03] += 1 ! INCREMENT COUNTRY_CODE ERROR !
    STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
  FI

```

```

FI
ELSE
! IT IS AT THIS POINT THAT OTHER CONTROL CODES
  WILL BE INCORPORATED INTO THE NETWORK LAYER. !
  DESTINATION := 'ER'
  STATTB [02] += 1 ! INCREMENT CONTROL_CODE ERROR !
  STATTB [00] += 1 ! INCREMENT LOCAL ERROR COUNT !
FI

```

END DET_DEST_FOUR

```

!*****!
!*****!
PROCEDURE DET_DEST_LL DETERMINES THE DESTINATION OF DATA TO A LOCAL HOST

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA RESIDING IN THE LOCAL TABLE.

INPUT - THE INPUT IS A TWO ASCII VALUE INDICATING THE TABLE LOCATION OF THE DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THEN THE HOST_CODE IS EXTRACTED TO DETERMINE WHICH LOCAL CHANNEL THE DATA SHOULD BE TRANSMITTED OUT OF.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('01','02','03', OR '04') TO WHICH THE DATA SHOULD BE TRANSFERRED TO.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_OUT.

- NOTES:
1. BYTE AND %0F WILL MASK OUT THE UPPER 4-BITS.
 2. BYTE AND %F0 WILL MASK OUT THE LOWER 4-BITS.
 3. BYTE / %01 WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.

DET_DEST_LL PROCEDURE (TABLE WORD)
RETURNS (DESTINATION WORD)

```

BITS_INTEGER
CONTROL_CODE_BYTE
HOST_CODE_INTEGER

```

```

HOST_CODE := 0
CONTROL_CODE := LCLCTB [LCLCNS+16] AND %F0
IF CONTROL_CODE = 00
THEN
  BITS := INTEGER (LCLCTB[LCLCNS+17] AND %0F)/%01
  HOST_CODE := BITS * 16
  BITS := INTEGER (LCLCTB[LCLCNS+18] AND %F0)/%10
  HOST_CODE := HOST_CODE + BITS
  IF (HOST_CODE>=0) ANDIF (HOST_CODE<=63) THEN
    DESTINATION := '01' FI
  IF (HOST_CODE>=64) ANDIF (HOST_CODE<=127) THEN
    DESTINATION := '02' FI
  IF (HOST_CODE>=128) ANDIF (HOST_CODE<=191) THEN
    DESTINATION := '03' FI
  IF (HOST_CODE>=192) ANDIF (HOST_CODE<=255) THEN
    DESTINATION := '04' FI
  ELSE
    DESTINATION := 'ER'
  STATTB [02] += 1 ! INCREMENT CONTROL_CODE ERROR

```

```

        STATTB [01] += 1 ! INCREMENT LOCAL ERROR COUNT !
    FI

```

```

END DET_DEST_LL

```

```

!*****!

```

```

!*****!
PROCEDURE DET_DEST_NL    DETERMINES THE DESTINATION OF DATA FROM NETWORK

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA COMING FROM THE NETWORK SIDE OF THE UNID TO A LOCAL HOST.

INPUT - THE INPUT IS A TWO ASCII VALUE INDICATING THE TABLE LOCATION OF THE DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE EXTRACTS THE DESTINATION_ADDRESS FROM THE SECOND BYTE OF THE DATA PACKET AND RETURNS THE CORRESPONDING DESTINATION.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION ('01','02','03', OR '04') OF THE DATA COMING FROM THE NETWORK TO A LOCAL HOST.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_OUT.

NOTES: 1. BYTE AND %07 WILL MASK OUT THE UPPER 5-BITS.

```

*****!

```

```

INTERNAL

```

```

DET_DEST_NL    PROCEDURE (TABLE WORD)
                RETURNS (DESTINATION WORD)

```

```

LOCAL

```


BITS INTEGER

ENTRY

```

BITS := 0
BITS := INTEGER (NTLCTB [NTLCNS+0] AND %07)
IF BITS
CASE 1 THEN DESTINATION := '01'
CASE 2 THEN DESTINATION := '02'
CASE 3 THEN DESTINATION := '03'
CASE 4 THEN DESTINATION := '04'
ELSE
DESTINATION := 'ER'
STATTB [01] += 1 ! INCREMENT LOCAL ERROR COUNT !
FI
```

END DET_DEST_NL

!*****

!*****

PROCEDURE LD_TAB_HSKP LOAD TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
BUFFER TABLE AFTER THE LOADING OF THE USER DATA FROM THE HOST.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT-EMPTY-BYTE POINTER BY ONE PACKET_SIZE,
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-EMPTY-BYTE
POINTER ADVANCED BY THE LENGTH OF A SINGLE PACKET.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN AND ROUTE_OUT.

NOTES - NONE.

INTERNAL
*****!

LD_TAB_HSKP PROCEDURE (TABLE WORD)
ENTRY

IF TABLE ! INDICATES WHICH TABLE NEEDS TO BE HOUSEKEEPT !

CASE '01' ! IF CALLED TO HSKP LOCAL CHANNEL-1 TABLE !
THEN
LC01NE := LC01NE + DATA_SIZE ! ADVANCE NEXT EMPTY PNTR !
IF LC01NE >= LC01SZ ! IF TABLE WRAP !
THEN
LC01NE := 0
FI

CASE '02' ! IF CALLED TO HSKP LOCAL CHANNEL-2 TABLE !
THEN
LC02NE := LC02NE + DATA_SIZE ! ADVANCE NEXT EMPTY PNTR !
IF LC02NE >= LC02SZ ! IF TABLE WRAP !
THEN
LC02NE := 0
FI

CASE '03' ! IF CALLED TO HSKP LOCAL CHANNEL-3 TABLE !
THEN
LC03NE := LC03NE + DATA_SIZE ! ADVANCE NEXT EMPTY PNTR !
IF LC03NE >= LC03SZ ! IF TABLE WRAP !
THEN
LC03NE := 0
FI

```

CASE '04' ! IF CALLED TO HSKP LOCAL CHANNEL-4 TABLE !
THEN
  LC04NE := LC04NE + DATA_SIZE ! ADVANCE NEXT EMPTY PNTR !
  IF LC04NE >= LC04SZ
  THEN
    LC04NE := 0
  FI

CASE 'LL' ! IF CALLED TO HSKP LOCAL TO LOCAL TABLE !
THEN
  LCCLNE := LCCLNE + DATA_SIZE ! ADVANCE NEXT EMPTY PNTR !
  IF LCCLNE >= LCCLSZ
  THEN
    LCCLNE := 0
  FI

CASE 'LN' ! IF CALLED TO HSKP LOCAL TO NETWORK TABLE !
THEN
  LCNTNE := LCNTNE + PACKET_SIZE ! ADVANCE NEXT EMPTY PNTR !
  IF LCNTNE >= LCNTSZ
  THEN
    LCNTNE := 0
  FI

  FI ! END OF IF TABLE CONDITION !

END LD_TAB_HSKP

!*****!
!*****!
PROCEDURE   SRVC_TAB_HSKP  SERVICE TABLE HOUSEKEEP

```

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT-BYTE-TO-BE-SERVED POINTER BY A PACKET_SIZE,
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-BYTE-TO-BE-SERVED
POINTER ADVANCED BY THE LENGTH OF A SINGLE PACKET.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE_IN AND ROUTE_OUT.

NOTES - NONE.

INTERNAL
*****!

SRVC_TAB_HSKP PROCEDURE (TABLE WORD)

ENTRY

IF TABLE ! DETERMINE WHICH TABLE MUST BE HOUSEKEEPPED !

CASE '01' ! IF CALLED TO HSKP LOCAL CHANNEL-1 TABLE !
THEN

LC01NS := LC01NS + DATA_SIZE ! ADVANCE NEXT SERVICE PNTR !

IF LC01NS > LC01SZ ! IF TABLE WRAP !

THEN

LC01NS := 0

! THEN SET PNTR TO ZERO !

FI

CASE '02' ! IF CALLED TO HSKP LOCAL CHANNEL-2 TABLE !

THEN

```

LC02NS := LC02NS + DATA_SIZE ! ADVANCE NEXT SERVICE PNTR !
IF LC02NS >= LC02SZ
THEN
    LC02NS := 0
    ! THEN SET PNTR TO ZERO !
FI

CASE '03' ! IF CALLED TO HSKP LOCAL CHANNEL-3 TABLE !
THEN
    LC03NS := LC03NS + DATA_SIZE ! ADVANCE NEXT SERVICE PNTR !
    IF LC03NS >= LC03SZ
    THEN
        LC03NS := 0
        ! THEN SET PNTR TO ZERO !
    FI

CASE '04' ! IF CALLED TO HSKP LOCAL CHANNEL-4 TABLE !
THEN
    LC04NS := LC04NS + DATA_SIZE ! ADVANCE NEXT SERVICE PNTR !
    IF LC04NS >= LC04SZ
    THEN
        LC04NS := 0
        ! THEN SET PNTR TO ZERO !
    FI

CASE 'LL' ! IF CALLED TO HSKP LOCAL TO LOCAL TABLE !
THEN
    LCLCNS := LCLCNS + DATA_SIZE ! ADVANCE NEXT SERVICE PNTR !
    IF LCLCNS >= LCLCSZ
    THEN
        LCLCNS := 0
        ! THEN SET PNTR TO ZERO !
    FI

CASE 'NL' ! IF CALLED TO HSKP NETWORK TO LOCAL TABLE !
THEN
    NTLCNS := NTLCNS + PACKET_SIZE !ADVANCE NEXT SERVICE PNTR!
    IF NTLCNS >= NTLCSZ
    THEN

```

```

      NTLCL := 0
      ! THEN SET PNTR TO ZERO !
      PI

```

FI ! END OF IF TABLE CONDITION !

END SRVC TAB_HSKP

```
*****  
! ***** TRNMIT_PKT TRANSMIT A PACKET *****  
*****  
PROCEDURE
```

THE PURPOSE OF THIS PROCEDURE IS TO SET UP THE DATA AND PORT ADDRESSES FOR PACKET TRANSMISSION, AND DRIVE BYTE TRANSMISSION UNTIL AN ENTIRE PACKET HAS BEEN SENT TO THE HOST.

INPUT - THE PACKET'S FIRST BYTE ADDRESS AND THE USART DATA PORT ADDRESS ARE INPUT TO TRNMIT_PKT.

PROCESSING - TWO GLOBAL VALUES, TDAADD (DATA ADDRESS) AND TPRADD (PORT ADDRESS), ARE LOADED WITH THE CORRECT INITIAL VALUES. THE PROCEDURE THEN LOOPS WITH BYTE TRANSMISSION VIA TRNMIT. THIS LOOP CONTINUES TO OUTPUT AND ADVANCE THE DATA ADDRESS UNTIL A FULL PACKET HAS BEEN TRANSMITTED.

OUTPUT - A PACKET_SIZE NUMBER OF BYTES ARE TRANSMITTED ON THE
LOCAL CHANNEL SPECIFIED BY PRTHDD.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_OUT.
ROUTE_OUT THEN CALLS PROCEDURE TRNMIT IN MODULE L.VINT_U2
FOR THE ACTUAL BYTE OUTPUT.

NOTES - NONE.

INTERNAL

TRNMIT_PKT PROCEDURE (SRCADD PBYTE, PRTADD BYTE)

LOCAL

IX INTEGER ! INDEX !

ENTRY

```

IX := 0
TDAADD := SRCADD
TPRADD := PRTADD
DO
    TRNMIT
    IX += 1
    TDAADD := INC TDAADD
    IF IX = DATA_SIZE ! TRANSMIT ONE DATA BLOCK !
    THEN
        EXIT
    FI
OD

```

END TRNMIT_PKT

```

!*****!
!*****!
PROCEDURE BUILD_I_PACKET
    INTO A DATA_PACKET FOR TRANSFER TO THE NETWORK
    SIDE OF THE UNID.

```

THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM THE HOST'S USER DATA DELIVERED TO ONE OF THE LOCAL INPUT TABLES INTO A DATA_PACKET TO BE PLACED IN THE LOCAL-TO-NETWORK TABLE. THIS PROCEDURE

ADDS ON THE FIVE HEADER BYTES, AS FOLLOWS:

1. 1-BYTE FOR DESTINATION ADDRESS.
2. 1-BYTE FOR SOURCE ADDRESS.
3. 1-BYTE FOR THE SEQUENCE NUMBER.
4. 1-BYTE FOR SPARE_01.
5. 1-BYTE FOR SPARE_02.

INPUT - THIS PROCEDURE RECEIVES A TWO CHARACTER ASCII VALUE INDICATING THE LOCAL INPUT TABLE WHERE THE INCOMING HOST DATA IS LOCATED.

PROCESSING - THE PROCEDURE BEGINS WITH THE PASSING OF THE TABLE WHERE THE HOST'S DATA IS LOCATED. THE SOURCE AND DESTINATION ADDRESS (SOURCE_ADDRESS, DESTINATION_ADDRESS) IS SUPPLIED BY THE DET_DEST PROCEDURE. FOR NOW, THE SEQUENCE NUMBER AND BOTH SPARE BYTES (SPARE_01, SPARE_02) ARE SET TO ZERO. IN THE FUTURE, THESE BYTES WILL REFLECT X.25 PROTOCOL.

OUTPUT - THIS PROCEDURE PLACES THE FIRST FIVE BYTES INTO THE LOCAL-TO-NETWORK TABLE BEFORE THE PACKET IS TRANSFERRED OVER TO THE NETWORK SIDE OF THE UNID.

INTERFACE - THE PROCEDURE IS CALLED FROM PROCEDURE-ROUTE_IN FOR THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.

NOTES: THE HEADER INFORMATION SUPPLIED IS FOR DATAGRAM SERVICE ONLY. ALSO, THE SEQUENCE AND SPARE BYTES ARE SET TO ZERO TO ALLOW THE UNID'S MINIMAL OPERATIONAL CAPABILITY. FOR FUTURE SOFTWARE ENHANCEMENTS THIS MUST BE MODIFIED.

*****!
INTERNAL

BUILD_I_PACKET PROCEDURE (TABLE WORD,
SOURCE_ADDRESS BYTE,
DESTINATION_ADDRESS BYTE)

LOCAL

```

DESTINATION_BYTE BYTE
SOURCE_BYTE BYTE
SEQUENCE_BYTE BYTE
SPARE_01_BYTE BYTE
SPARE_02_BYTE BYTE

```

ENTRY

IF TABLE ! OBTAIN WHICH TABLE THE HOST DATA IS LOCATED. !

CASE '01' ! THE HOST DATA IS LOCATED IN LOCAL TABLE-1 !
 THEN

```

DESTINATION_BYTE := DESTINATION_ADDRESS
SOURCE_BYTE      := SOURCE_ADDRESS
SEQUENCE_BYTE    := 0
SPARE_01_BYTE    := 0
SPARE_02_BYTE    := 0
LCNTTB [LCNTNE+0] := DESTINATION_BYTE
LCNTTB [LCNTNE+1] := SOURCE_BYTE
LCNTTB [LCNTNE+2] := SEQUENCE_BYTE
LCNTTB [LCNTNE+3] := SPARE_01_BYTE
LCNTTB [LCNTNE+4] := SPARE_02_BYTE

```

CASE '02' ! THE HOST DATA IS LOCATED IN LOCAL TABLE-2 !
 THEN

```

DESTINATION_BYTE := DESTINATION_ADDRESS
SOURCE_BYTE      := SOURCE_ADDRESS
SEQUENCE_BYTE    := 0
SPARE_01_BYTE    := 0
SPARE_02_BYTE    := 0
LCNTTB [LCNTNE+0] := DESTINATION_BYTE
LCNTTB [LCNTNE+1] := SOURCE_BYTE

```

```

LCNTTB [LCNTNE+2] := SEQUENCE_BYTE
LCNTTB [LCNTNE+3] := SPARE_01_BYTE
LCNTTB [LCNTNE+4] := SPARE_02_BYTE

```

```

CASE '03' ! THE HOST DATA IS LOCATED IN LOCAL TABLE-3 !
THEN

```

```

    DESTINATION_BYTE := DESTINATION_ADDRESS
    SOURCE_BYTE      := SOURCE_ADDRESS
    SEQUENCE_BYTE    := 0
    SPARE_01_BYTE    := 0
    SPARE_02_BYTE    := 0
    LCNTTB [LCNTNE+0] := DESTINATION_BYTE
    LCNTTB [LCNTNE+1] := SOURCE_BYTE
    LCNTTB [LCNTNE+2] := SEQUENCE_BYTE
    LCNTTB [LCNTNE+3] := SPARE_01_BYTE
    LCNTTB [LCNTNE+4] := SPARE_02_BYTE

```

```

CASE '04' ! THE HOST DATA IS LOCATED IN LOCAL TABLE-4 !
THEN

```

```

    DESTINATION_BYTE := DESTINATION_ADDRESS
    SOURCE_BYTE      := SOURCE_ADDRESS
    SEQUENCE_BYTE    := 0
    SPARE_01_BYTE    := 0
    SPARE_02_BYTE    := 0
    LCNTTB [LCNTNE+0] := DESTINATION_BYTE
    LCNTTB [LCNTNE+1] := SOURCE_BYTE
    LCNTTB [LCNTNE+2] := SEQUENCE_BYTE
    LCNTTB [LCNTNE+3] := SPARE_01_BYTE
    LCNTTB [LCNTNE+4] := SPARE_02_BYTE

```

```

ELSE

```

```

    STATTB [00] += 1      ! IF IMPROPER TABLE INCREMENT THEN !
                        ! SENT AND ERROR TO THE ERROR TABLE !

```

```

FI ! END OF IF TABLE CONDITION !

```


INTERNAL

ROUTE_IN PROCEDURE

LOCAL

SOURCE_ADDRESS BYTE
DESTINATION_ADDRESS BYTE

ENTRY

SOURCE_ADDRESS := 00
DESTINATION_ADDRESS := 00

```
IF ((LC01NE-LC01NS) >= DATA_SIZE) ! IF CHANNEL-1 READY !
ORIF (LC01NS > LC01NE) ! OR MORE DATA IN TABLE !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_6[0],39)
  ! SEND TEST_POINT_6 MESSAGE TO LOCAL MONITOR !
  DESTINATION,DESTINATION_ADDRESS := DET_DEST_ONE('01')
```

IF DESTINATION

```
CASE 'LL' ! DATA IS LOCAL-TO-LOCAL TRANSFER !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_7[0],41)
  ! SEND TEST_POINT_7 MESSAGE TO LOCAL MONITOR !
  MOVSEQ( LC01TB[LC01NS], LCLCTB[LCLCNE],DATA_SIZE)
  ! MOVE THE DATA TO THE LOCAL-TO-LOCAL TABLE !
  LD_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !
```

```
CASE 'LN' ! DATA IS LOCAL-TO-NETWORK TRANSFER !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_8[0],41)
```

```

! SEND TEST_POINT_8 MESSAGE TO LOCAL MONITOR !
SOURCE_ADDRESS := %21
BUILD_I_PACKET('01',SOURCE_ADDRESS,DESTINATION_ADDRESS)
! BUILD A DATA PACKET BEFORE TRANSFER TO NETWORK !
MOVSEQ( LC01TB[LC01NS], LCNTTB[LCNTNE+5],DATA_SIZE)
! MOVE DATA TO NETWORK TABLE !
LD_TAB_HSKP('LN')
! HOUSEKEEP THE LOCAL-TO-NETWORK TABLE !

ELSE
  SNDSEQ(CONCMD,CONDAT, TP_9[0],55)
  ! SEND TEST_POINT_9 MESSAGE TO LOCAL MONITOR !
  STATTB[L_RI_DEST_ERR] += 1 ! COUNT AN ERROR !

FI ! END OF IF DESTINATION CONDITION !

SRVC_TAB_HSKP('01')
!HOUSEKEEP LOCAL CHANNEL-1 TABLE !

FI ! END OF IF (LC01NE-LC01NS) CONDITION !

IF ((LC02NE-LC02NS) >= DATA_SIZE) ! IF CHANNEL-2 READY !
ORIF (LC02NS > LC02NE) ! OR MORE DATA IN TABLE !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_10[0],40)
  ! SEND TEST_POINT_10 MESSAGE TO LOCAL MONITOR !
  DESTINATION,DESTINATION_ADDRESS := DET_DEST_TWO('02')

IF DESTINATION

CASE 'LL' ! DATA IS LOCAL-TO-LOCAL TRANSFER !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_11[0],42)
  ! SEND TEST_POINT_11 MESSAGE TO LOCAL MONITOR !
  MOVSEQ( LC02TB[LC02NS], LCLCTB[LCLCNE],DATA_SIZE)

```

```

      ! MOVE THE DATA TO LOCAL-TO-LOCAL TABLE !
      LD_TAB_HSKP('LL')
      ! HOUSEKEEP LOCAL-TO-LOCAL TABLE !

      CASE 'LN' ! DATA IS LOCAL-TO-NETWORK TRANSFER !
      THEN
        SNDSEQ(CONCMD,CONDAT, TP_12[0],42)
        ! SEND TEST_POINT_12 MESSAGE TO LOCAL MONITOR !
        SOURCE_ADDRESS := %22
        BUILD_I_PACKET('02',SOURCE_ADDRESS,DESTINATION_ADDRESS)
        ! BUILD A DATA PACKET BEFORE TRANSFER TO NETWORK !
        MOVSEQ( LC02TB[LC02NS], LCNTTB[LCNTNE+5],DATA_SIZE)
        ! MOVE DATA TO THE NETWORK SIDE !
        LD_TAB_HSKP('LN')
        ! HOUSEKEEP THE LOCAL-TO-NETWORK TABLE !

      ELSE
        SNDSEQ(CONCMD,CONDAT, TP_13[0],55)
        ! SEND TEST_POINT_13 MESSAGE TO LOCAL MONITOR !
        STATTB[L_RI_DEST_ERR] += 1 ! COUNT AN ERROR !

      FI ! END OF IF DESTINATION CONDITION !

      SRVC_TAB_HSKP('02')
      ! HOUSEKEEP LOCAL CHANNEL-2 TABLE !

      FI ! END OF IF (LC02NE-LC02NS) CONDITION !

      IF ((LC03NE-LC03NS) >= DATA_SIZE) ! IF CHANNEL-3 READY !
      ORIF (LC03NS > LC03NE) ! OR MORE DATA IN TABLE !
      THEN
        SNDSEQ(CONCMD,CONDAT, TP_14[0],40)
        ! SEND TEST_POINT_14 MESSAGE TO LOCAL MONITOR !
        DESTINATION,DESTINATION_ADDRESS := DET_DEST_THREE('03')

```

```

IF DESTINATION

CASE 'LL' ! DATA IS LOCAL-TO-LOCAL TRANSFER !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_15[0],42)
  ! SEND TEST_POINT_15 MESSAGE TO LOCAL MONITOR !
  MOVSEQ( LC03TB[LC03NS], LCLCTB[LCLCNE],DATA_SIZE)
  ! MOVE DATA TO LOCAL-TO-LOCAL TABLE !
  LD_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

CASE 'LN' ! DATA IS LOCAL TO NETWORK TRANSFER !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_16[0],42)
  ! SEND TEST_POINT_16 MESSAGE TO LOCAL MONITOR !
  SOURCE_ADDRESS := %23
  BUILD_I_PACKET('03',SOURCE_ADDRESS,DESTINATION_ADDRESS)
  ! BUILD A DATA PACKET BEFORE TRANSFER TO NETWORK !
  MOVSEQ( LC03TB[LC03NS], LCNTTB[LCNTNE+5],DATA_SIZE)
  ! MOVE DATA TO THE LOCAL-TO-NETWORK TABLE !
  LD_TAB_HSKP('LN')
  ! HOUSEKEEP THE LOCAL-TO-NETWORK TABLE !

ELSE
  SNDSEQ(CONCMD,CONDAT, TP_17[0],55)
  ! SEND TEST_POINT_17 MESSAGE TO LOCAL MONITOR !
  STATTB[L_RI_DEST_ERR] += 1 ! COUNT AN ERROR !

FI ! END OF IF DESTINATION CONDITION !

SRVC_TAB_HSKP('03')
! HOUSEKEEP LOCAL CHANNEL-3 TABLE !

FI ! END OF IF (LC03NE-LC03NS) CONDITION !

```

```

IF ((LC04NE-LC04NS) >= DATA_SIZE) ! IF LOCAL CHANNEL-4 READY !
ORIF (LC04NS > LC04NE) ! OR MORE DATA IN TABLE !
THEN
    SNDSEQ(CONCMD,CONDAT, TP_18[0],40)
    ! SEND TEST_POINT_18 MESSAGE TO LOCAL MONITOR !
    DESTINATION,DESTINATION_ADDRESS := DET_DEST_FOUR('04')
IF DESTINATION
CASE 'LL' ! DATA IS LOCAL-TO-LOCAL TRANSFER !
THEN
    SNDSEQ(CONCMD,CONDAT, TP_19[0],42)
    ! SEND TEST_POINT_19 MESSAGE TO LOCAL MONITOR !
    MOVSEQ( LC04TB[LC04NS], LCLCTB[LCLCNE],DATA_SIZE)
    ! MOVE DATA TO LOCAL-TO-LOCAL TABLE !
    LD_TAB_HSKP('LL')
    ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !
CASE 'LN' ! DATA IS LOCAL-TO-NETWORK TRANSFER !
THEN
    SNDSEQ(CONCMD,CONDAT, TP_20[0],42)
    ! SEND TEST_POINT_20 MESSAGE TO LOCAL MONITOR !
    SOURCE_ADDRESS := %24
    BUILD_I_PACKET('04',SOURCE_ADDRESS,DESTINATION_ADDRESS)
    ! BUILD A DATA PACKET BEFORE TRANSFER TO NETWORK !
    MOVSEQ( LC04TB[LC04NS], LCNTTB[LCNTNE+5],DATA_SIZE)
    ! MOVE DATA TO LOCAL-TO-NETWORK TABLE !
    LD_TAB_HSKP('LN')
    ! HOUSEKEEP LOCAL-TO-NETWORK TABLE !
ELSE
    SNDSEQ(CONCMD,CONDAT, TP_21[0],55)
    ! SEND TEST_POINT_21 MESSAGE TO LOCAL MONITOR !
    STATTB[L_RI_DEST_ERR] += 1 ! COUNT AN ERROR !

```


FI ! END OF IF DESTINATION CONDITION !

SRVC_TAB_HSKP('04')

! HOUSEKEEP LOCAL CHANNEL-4 TABLE !

FI ! END OF IF (LC04NE-LC04NS) CONDITION !

END ROUTE_IN

!*****

!*****

PROCEDURE ROUTE_OUT ROUTE OUT EITHER A DATA_PACKET OR HOST DATA

THE PURPOSE OF THIS PROCEDURE IS TO ROUTE PACKETS FROM
THE LOCAL-TO-LOCAL AND NETWORK-TO-LOCAL TABLES TO THE
CORRECT OUTPUT CHANNEL.

INPUT - DATA PACKETS ARE ROUTED VIA EVALUATION OF LCLCTB AND NTLCTB
POINTERS AND INTERNAL PACKET ROUTING INFORMATION.

PROCESSING - THE PROCEDURE CHECKS EACH INPUT BUFFER'S POINTERS FOR
PACKET ARRIVAL. IF A PACKET IS READY, THE DESTINATION IS
DETERMINED VIA PROC DET_DEST, AND TRANSMITTED VIA PROC TRNMIT_PKT.
THE TABLE THAT WAS SERVICED (PACKET REMOVED) HAS ITS POINTERS
HOUSEKEPT BY SRVC_TAB_HSKP.
WHEN THE FRAME IS TRANSMITTED TO THE HOSTS FROM THE NET-TO-LOCAL
BUFFER TABLE, THE FIRST TWO BYTES OF HEADER INFORMATION IS STRIPPED
OFF BEFORE TRANSMISSION.

OUTPUT - A PACKET OF DATA IS TRANSMITTED TO A LOCAL CHANNEL.

INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY PROCEDURE
L.MAIN_U2.

NOTES - NONE.
 *****!

INTERNAL

ROUTE_OUT PROCEDURE

ENTRY

```

IF ((LCLCNE-LCLCNS) >= DATA_SIZE) ! IF LOCAL TO LOCAL READY !
ORIF (LCLCNS > LCLCNE) ! OR MORE DATA IN TABLE !
THEN
  SNDSEQ(CONCMD, CONDAT, TP_23[0], 35)
  ! SEND TEST_POINT_23 MESSAGE TO LOCAL MONITOR !
  DESTINATION := DET_DEST_LL('LL')

```

IF DESTINATION

```

CASE '01' ! DATA IS DESTINED FOR LOCAL CHANNEL-1 !
THEN
  SNDSEQ(CONCMD, CONDAT, TP_24[0], 52)
  ! SEND TEST_POINT_24 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( LCLCTB[LCLCNS], U01DAT)
  ! TRANSMIT DATA TO LOCAL PORT-1 !
  SRVC_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

```

```

CASE '02' ! DATA IS DESTINED FOR LOCAL CHANNEL-2 !
THEN
  SNDSEQ(CONCMD, CONDAT, TP_25[0], 52)
  ! SEND TEST_POINT_25 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( LCLCTB[LCLCNS], U02DAT)
  ! TRANSMIT DATA TO LOCAL PORT-2 !
  SRVC_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

```

```

CASE '03' ! DATA IS DESTINED FOR LOCAL CHANNEL-3 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_26[0],52)
  ! SEND TEST_POINT_26 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( LCLCTB[LCLCNS],U03DAT)
  ! TRANSMIT DATA TO LOCAL PORT-3 !
  SRVC_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

CASE '04' ! DATA IS DESTINED FOR LOCAL CHANNEL-4 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_27[0],52)
  ! SEND TEST_POINT_27 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( LCLCTB[LCLCNS],U04DAT)
  ! TRANSMIT DATA TO LOCAL PORT-4 !
  SRVC_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

ELSE
  SNDSEQ(CONCMD,CONDAT, TP_28[0],48)
  ! SEND TEST_POINT_28 MESSAGE TO LOCAL MONITOR !
  STATB[L_RO_DEST_ERR] += 1 ! COUNT AN ERROR !
  SRVC_TAB_HSKP('LL')
  ! HOUSEKEEP THE LOCAL-TO-LOCAL TABLE !

FI ! END OF IF DESTINATION CONDITION !

FI ! END OF IF (LCLCNE-LCLCNS) CONDITION !

IF ((NTLCNE-NTLCNS) >= PACKET_SIZE) ! IF NET TO LOCAL READY !
ORIF (NTLCNS > NTLCNE) ! OR MORE DATA IN TABLE !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_29[0],35)
  ! SEND TEST_POINT_29 MESSAGE TO LOCAL MONITOR !

```

```

DESTINATION := DET_DEST_NL('NL')

IF DESTINATION

CASE '01' ! DATA IS DESTINED FOR LOCAL CHANNEL-1 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_30[0],52)
  ! SEND TEST_POINT_30 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( NTLCTB[NTLCNS+5],U01DAT)
  ! TRANSMIT DATA MINUS HEADER TO LOCAL PORT-1 !
  SRVC_TAB_HSKP('NL')
  ! HOUSEKEEP THE NETWORK-TO-LOCAL TABLE !

CASE '02' ! DATA IS DESTINED FOR LOCAL CHANNEL-2 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_31[0],52)
  ! SEND TEST_POINT_31 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( NTLCTB[NTLCNS+5],U02DAT)
  ! TRANSMIT DATA MINUS HEADER TO LOCAL PORT-2 !
  SRVC_TAB_HSKP('NL')
  ! HOUSEKEEP NETWORK-TO-LOCAL TABLE !

CASE '03' ! DATA IS DESTINED FOR LOCAL CHANNEL-3 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_32[0],52)
  ! SEND TEST_POINT_32 MESSAGE TO LOCAL MONITOR !
  TRNMIT_PKT( NTLCTB[NTLCNS+5],U03DAT)
  ! TRANSMIT DATA MINUS HEADER TO LOCAL PORT-3 !
  SRVC_TAB_HSKP('NL')
  ! HOUSEKEEP NETWORK-TO-LOCAL TABLE !

CASE '04' ! DATA IS DESTINED FOR LOCAL CHANNEL-4 !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_33[0],52)
  ! SEND TEST_POINT_33 MESSAGE TO LOCAL MONITOR !

```

```

TRNMIT_PKT( NTLCTB[NTLCNS+5],U04DAT)
! TRANSMIT DATA MINUS HEADER TO LOCAL PORT-4 !
SRVC_TAB_HSKP('NL')
! HOUSEKEEP NETWORK-TO-LOCAL TABLE !

```

```

ELSE
  SNDSEQ(CONCMD,CONDAT, TP_34[0],48)
  ! SEND TEST_POINT_34 MESSAGE TO LOCAL MONITOR !
  STATB[L_RO_DEST_ERR] += 1 ! COUNT AN ERROR !
  SRVC_TAB_HSKP('NL')
  ! HOUSEKEEP NETWORK-TO-LOCAL TABLE !

```

```

FI ! END OF IF DESTINATION CONDITION !

```

```

FI ! END OF IF (NTLCNE-NTLCNS) CONDITION !

```

```

END ROUTE_OUT

```

```

!*****!

```

```

GLOBAL

```

```

!*****!

```

```

PROCEDURE MAIN PROCEDURE FOR MAIN LINE DRIVER OF LOCAL OS

```

```

THE PURPOSE OF THIS PROCEDURE IS TO PROVIDE THE MAIN LINE
OF PROCESSING FOR THE LOCAL OPERATING SYSTEM (L.OS).

```

```

INPUT - NONE.

```

```

PROCESSING - THIS PROCEDURE SENDS A HEADER TO THE LOCAL MONITOR
CONSOLE, INITIALIZES THE LOCAL BUFFERS AND STATUS BUFFER
VIA INIT_L_TAB, INITIALIZES THE SHARED BUFFERS VIA INIT_U_SHTAB,
USES PROCEDURE-INVINT TO INITIALIZE I/O VECTOR INTERRUPTS,
AND LOOPS ENDLESSLY ROUTING PACKETS IN AND OUT VIA PROCEDURES

```

ROUTE_IN AND ROUTE_OUT.

OUTPUT - A START UP MESSAGE IS SENT TO THE LOCAL MONITOR UPON
START UP.

INTERFACE - THIS PROCEDURE IS THE INITIAL ENTRY POINT FOR THE
LOCAL OPERATING SYSTEM (L.OS).
IT OPERATES THROUGH SINGLE CALLS TO SNDSEQ, INIT_L_TAB, INIT_U_SHTAB,
AND INVINT; AND REPETITIVE CALLS TO ROUTE_IN AND ROUTE_OUT.

NOTES - NONE.

MAIN PROCEDURE
*****!

ENTRY

SNDSEQ(CONCMD,CONDAT, STARTUP_HDR[0],50)
! SEND STARUP HEADER TO THE LOCAL MONITOR !

SNDSEQ(CONCMD,CONDAT, TP_1[0],37)
! SEND TEST_POINT_1 MESSAGE TO THE LOCAL MONITOR !

INIT_L_TAB
! INITIALIZE ALL THE LOCAL TABLES !

SNDSEQ(CONCMD,CONDAT, TP_2[0],39)
! SEND TEST_POINT_2 MESSAGE TO THE LOCAL MONITOR !

INIT_U_SHTAB
! INITIALIZE ALL UNID SHARED TABLES !

SNDSEQ(CONCMD,CONDAT, TP_3[0],33)
! SEND TEST_POINT_3 MESSAGE TO THE LOCAL MONITOR !

INVINT

```

! INITIALIZE ALL THE INPUT/OUTPUT VECTOR INTERRUPTS !
SNDSEQ(CONCMD,CONDAT, TP_4[0],40)
! SEND TEST_POINT_4 MESSAGE TO THE LOCAL MONITOR !

DO
! START OF ENDLESS LOOP TO ROUTE DATA TO AND FROM THE UNID !

SNDSEQ(CONCMD,CONDAT, TP_5[0],35)
! SEND TEST_POINT_5 MESSAGE TO THE LOCAL MONITOR !

ROUTE_IN
! ROUTE IN ANY DATA FROM THE HOST OR NETWORK SIDE OF UNID !

SNDSEQ(CONCMD,CONDAT, TP_22[0],37)
! SEND TEST_POINT_22 MESSAGE TO THE LOCAL MONITOR !

ROUTE_OUT
! ROUTE OUT ANY DATA TO HOST OR NETWORK SIDE OF UNID !

SNDSEQ(CONCMD,CONDAT, TP_35[0],39)
! SEND TEST_POINT_35 MESSAGE TO THE LOCAL MONITOR !

OD

SNDSEQ(CONCMD,CONDAT, TP_36[0],44)
! SEND TEST_POINT_36 MESSAGE TO THE LOCAL MONITOR !

END MAIN

END MAIN

! *****!

```

```
*****  
! *****  
MODULE      L.TAB_U2    L.OS TABLES  
*****  
DATE ORIGINATED: 26 OCT 82  
DATE LAST MODIFIED:
```

NOTES: 1. ALL LOCAL TABLES ARE 128-BYTES IN LENGTH.

L TAB U2 MODULE *****!

```
GLOBAL
LC01TB ARRAY [DATA_TABLE_SIZE BYTE]
LC01NS INTEGER
LC01NE INTEGER
LC01SZ INTEGER

LC02TB ARRAY [DATA_TABLE_SIZE BYTE]
LC02NS INTEGER
LC02NE INTEGER
LC02SZ INTEGER

LC03TB ARRAY [DATA_TABLE_SIZE BYTE]
LC03NS INTEGER
LC03NE INTEGER
LC03SZ INTEGER
```



```

LC04TB ARRAY [DATA_TABLE_SIZE BYTE]
LC04NS INTEGER
LC04NE INTEGER
LC04SZ INTEGER

LCLCTB ARRAY [DATA_TABLE_SIZE BYTE]
LCLCNS INTEGER
LCLCNE INTEGER
LCLCSZ INTEGER

```

GLOBAL

```

!*****
*****
PROCEDURE INIT_L_TAB PROCEDURE TO INITIALIZE LOCAL DATA TABLES
*****
*****

```

THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE DATA TABLES USED BY THE LOCAL OPERATING SYSTEM (L.OS).

INPUT - NONE.

PROCESSING - THE PROCEDURE INITIALIZES THE FOUR LOCAL CHANNEL INPUT AND LOCAL-TO-LOCAL TABLES BY SETTING THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO A MULTIPLE OF PACKET_SIZE.

OUTPUT - THE TABLE POINTERS AS NOTED UNDER PROCESSING ARE MODIFIED.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-L.MAIN_U2.

NOTES ~ 1. ALL LOCAL TABLES ARE 128-BYTES IN LENGTH.

```

*****
INIT_L_TAB PROCEDURE
*****!

```

ENTRY

```

! INITIALIZE MEMORY TABLE INFORMATION
xxxxns - NEXT BYTE TO BE SERVICED
xxxxne - NEXT EMPTY BYTE
xxxxsz - SIZE OF TABLE
!

! INITIALIZE LOCAL CHANNEL-1 INPUT TABLE !
LC01ns := 0
LC01ne := 0
LC01sz := DATA_TABLE_SIZE

! INITIALIZE LOCAL CHANNEL-2 INPUT TABLE !
LC02ns := 0
LC02ne := 0
LC02sz := DATA_TABLE_SIZE

! INITIALIZE LOCAL CHANNEL-3 INPUT TABLE !
LC03ns := 0
LC03ne := 0
LC03sz := DATA_TABLE_SIZE

! INITIALIZE LOCAL CHANNEL-4 INPUT TABLE !
LC04ns := 0
LC04ne := 0
LC04sz := DATA_TABLE_SIZE

! INITIALIZE LOCAL-TO-LOCAL TABLE !
LCCLns := 0
LCCLne := 0
LCCLsz := DATA_TABLE_SIZE

END INIT_L_TAB

```

!*****!

END L_TAB_U2

!*****!

!*****!


```

; THE SECOND LOCATION TO BE ORG'D IS THE START
; OF THE PROCEDURES THAT FOLLOW THE TABLE. THESE PROCS
; MUST BE LOCATED AT A POINT BEYOND THE END OF IOVCTB.
;
; INPUT - THE ADDRESS FOR RETURNING TO THE CALLING
; PROCEDURE IS LOCATED ON THE TOP OF THE STACK.
;
; PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE IX REG FOR
; NORMALIZATION AT THE RETURN. EACH LOCAL CARD USART IS
; THEN INITIALIZED. THIS INITIALIZATION IS ACCOMPLISHED
; BY PASSING A DATA LIST ADDRESS THROUGH REG SET HL FOR
; USE BY PROCEDURE-INIURT. PROCEDURE INIURT WILL THEN OUTPUT THE
; COMMANDS ON THE DATA LIST TO THE ADDRESSES ON THE DATA
; LIST.
;
; INTERRUPT REGISTER (I) INITIALIZATION IS NEXT
; WITH THE I REG BEING LOADED WITH THE ADDRESS OF THE I/O
; VECTOR TABLE (IOVCTB). THIS TABLE MUST BE LOCATED ON AN
; EVEN 100 HEX MEMORY BOUNDARY (1600, 1700, ETC.). WHEN AN
; INTERRUPT IS IDENTIFIED BY THE PIC, THE I REG SUPPLIES
; THE HIGH 8 BITS AND THE PIC SUPPLIES THE LOW 8 BITS OF THE
; ADDRESS TO THE I/O CONTROLLER THAT WILL SERVICE THE INTERRUPT.
; WITH THE IOVCTB TABLE ON AN EVEN MEMORY BOUNDARY, ONLY THE
; 8 HIGH BITS ARE REQUIRED TO BE LOADED AS THE LOWER BITS
; ARE ALL ZEROS.
;
; PIC INITIALIZATION FOLLOWS WITH INTERRUPT MODE
; SET TO 2 AND INTERRUPTS ENABLED. AT THIS POINT, THE UNID
; IS CONFIGURED FOR INTERRUPT DRIVEN COMMUNICATIONS ON THE
; LOCAL SIDE.
;
; FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
; RECOVERED, AND A RETURN EXECUTED.
;
; OUTPUT - PROCEDURE INIURT IS PASSED DATA LIST ADDRESSES THROUGH
; REG SET HL. ADDITIONALLY, THE I REG IS SET WITH THE HIGH
; 8 BITS OF IOVCTB ADDRESS, THE PIC PRIORITY COMMAND IS
; OUTPUT ON THE PIC ADDRESS PORT, AND INTERRUPTS ENABLED IN

```

```

;
;
;INTERFACE - THIS PROCEDURE IS CALLED FROM PROCEDURE-L.MAIN_U2, THE MAIN LINE
;
;
; THIS PROCEDURE ALSO CALLS PROCEDURE INIURT AND PASSES A
; DATA LIST ADDRESS TO INIURT VIA THE HL REG SET.
;
; THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
; COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
; PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
; AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
; OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
; PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
;
;NOTES - NONE.
;
;*****
; ***** NOTE *****
;
; THIS MODULE MUST BE ORG'D IN TWO AREAS:
; THEREFORE WHEN PLINKING BE SURE THAT
; THIS PORTION BEGINS AT AN EVEN BOUNDARY
; LAYER SUCH AS:
; PLINK $=5000
; IMPORTANT INFO ABOUT THIS ORG IS
; IN THE PROC DOCUMENTATION ABOVE.
; ***** NOTE *****
; IO VECTOR ADDRESS TABLE
; USART 4 TRANSMIT CALLS URTT04
; USART 3 TRANSMIT CALLS URTT03
; USART 2 TRANSMIT CALLS URTT02
; USART 1 TRANSMIT CALLS URTT01
; USART 4 RECEIVE CALLS URTR04
; USART 3 RECEIVE CALLS URTR03
; USART 2 RECEIVE CALLS URTR02
; USART 1 RECEIVE CALLS URTR01

```

ORG 0000H

```

IOVCTB
URT04T DEFW URTTRN
URT03T DEFW URTTRN
URT02T DEFW URTTRN
URT01T DEFW URTTRN
URT04R DEFW URTR04
URT03R DEFW URTR03
URT02R DEFW URTR02
URT01R DEFW URTR01

```

```

; **** NOTE ****
;
; THIS MODULE IS ORG'D IN TWO AREAS:
; THE CODE, AND THE TABLE IOVCTB.
; IMPORTANT INFO ABOUT THIS ORG IS
; IN THE PROCEDURE DOCUMENTATION ABOVE.
; **** NOTE ****
;
; PROCEDURE TO INITIALIZE VECTOR INTERRUPT MODE
; STORE IX FOR RETURN
;
; LD LOC OF USART 1 DATA LIST
; INITIALIZE USART 1
; LD LOC OF USART 2 DATA LIST
; INITIALIZE USART 2
; LD LOC OF USART 3 DATA LIST
; INITIALIZE USART 3
; LD LOC OF USART 4 DATA LIST
; INITIALIZE USART 4
;
; LD ADD OF VECTOR ADDRESS TABLE
; LD HIGH 8 BITS OF ADD INTO I REG FOR BASE

ORG 0020H

; LD A, (PICCMD)
; OUT (PICADD), A
; SEND COMMAND

IM 2
EI

; SET INTERRUPT MODE TO VECTOR ADD MODE
; ENABLE INTERRUPTS

POP IX
POP HL
; RESTORE IX
; RECOVER RETURN ADDRESS

JP (HL)
; RETURN

```

INVINT:

```

PUSH IX

LD HL,URT01L
CALL INIURT
LD HL,URT02L
CALL INIURT
LD HL,URT03L
CALL INIURT
LD HL,URT04L
CALL INIURT

```

```

LD HL,IOVCTB
LD A,H
LD I,A

```

```

LD A, (PICCMD)
OUT (PICADD), A
; SEND COMMAND

IM 2
EI

; SET INTERRUPT MODE TO VECTOR ADD MODE
; ENABLE INTERRUPTS

POP IX
POP HL
; RESTORE IX
; RECOVER RETURN ADDRESS

JP (HL)
; RETURN

```

;EQUATES

```

PICADD EQU 018H ; PIC PORT ADDRESS
PICCMD EQU 00001000B ; ALLOW ANY INTERRUPT, NO PRIORITY

```

```

;*****

```

```

*EJECT

```

```

;*****

```

```

;*****

```

```

;PROCEDURE INIURT INITIALIZE LOCAL CARD USARTS

```

```

;
; THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE 2651
; USARTS ON THE UNID 2 LOCAL BOARD.

```

```

;INPUT - A DATA LIST ADDRESS IS PASSED TO INIURT VIA REG SET
; HL. THIS ADDRESS WILL IDENTIFY THE DATA TO BE USED BY INIURT.
; THE DATA LISTS ARE DEFINED IMMEDIATELY FOLLOWING THIS PROCEDURE.

```

```

;PROCESSING - A SEQUENCE OF COMMANDS TO THE USART AND ITS ASSOCIATED
; COUNTER TIMER CIRCUIT (CTC) ARE REQUIRED FOR INITIALIZATION.
; PROC INIURT SENDS A USART COMMAND WORD FOLLOWED BY TWO MODE
; WORDS. THE CTC IS THEN ADDRESSED AND A MODE WORD FOLLOWED BY
; A BAUD RATE PRESCALER CONSTANT IS OUTPUT. THESE ADDRESSES AND
; THE DATA TO BE OUTPUT ARE PRESET IN A TABLE THAT IS IDENTIFIED
; BY THE INPUT PARAMETER.

```

```

;OUTPUT - THE USART RECEIVES ONE COMMAND WORD FOLLOWED BY TWO
; MODE COMMANDS. THE CTC ASSOCIATED WITH THE USART RECEIVES
; A COMMAND WORD FOLLOWED BY A BAUD RATE CONSTANT.

```

```

;INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-INIURT. A DATA LIST
; ADDRESS IS PASSED VIA THE HL REG SET. THIS ADDRESS
; IDENTIFIES THE STARTING LOCATION OF THE DATA AND PORT

```



```

;
; ADDRESSES TO BE USED.
;
; NOTES -
; 1. INFORMATION CONCERNING THE 2651 USART AND THE CTC
; CAN BE OBTAINED IN:
; KANE, JERRY AND ADAM OSBORNE.
; AN INTRODUCTION TO MICROCOMPUTERS
; OSBORNE/MCGRAW-HILL, 1978.
;
; *****
; SUBROUTINE TO INITIALIZE A USART
; LD COMMAND PORT ADDRESS INTO C
; *****
INIURT:
LD A,3
ADD A,(HL)
LD C,A

INC HL
INC HL
INC HL
INC HL

; SET HL AT COMMAND LOC IN LIST

OUTI
; SEND COMMAND

DEC C
; SET C TO MODE ADDRESS

OUTI
OUTI
; SEND MODE COMMANDS

LD C,(HL)
INC HL
; SET C TO CTC CH ADDRESS
; SET HL TO CTC MODE COMMAND

OUTI
OUTI
; SEND MODE COMMAND AND PRESCALER

RET
; RETURN

;DEFINES

```


URT017	DEFB	11H	7 - 0: NOT USED
URT018	DEFB	01000101B	; ASSOC CTC - CTC1, CH1 ADDRESS
			; CTC MODE COMMAND
			; BIT 0 - 1: CONTROL WORD
			; 1 - 0: RESET-CONTINUED OPERATION
			; 2 - 1: TIME CONSTANT NEXT WORD
			; 3 - 0: AUTOMATIC TRIGGER
			; 4 - 0: SELECT (-)EDGE
			; 5 - 0: PRESCALER OF 16
			; 6 - 1: COUNTER MODE
			; 7 - 0: DISABLE INTERRUPTS
URT019	DEFB	04H	; CTC TIME CNST PRESCALER FOR 19.2 KBAUD
URT02L			; USART 2 INITIALIZATION DATA LIST
URT020	DEFB	04H	; DATA ADDRESS
URT021	DEFB	05H	; STATUS ADDRESS
URT022	DEFB	06H	; MODE ADDRESS
URT023	DEFB	07H	; COMMAND ADDRESS
URT024	DEFB	00100110B	; PROGRAM USART COMMAND REGISTER
			; BIT 0 - 0: DISABLE TRANSMIT
			; 1 - 1: FORCE NOT(DTR) LOW
			; 2 - 1: DISABLE RECEIVE
			; 3 - 0: ASYNCH-NORMAL
			; 4 - 0: RESET ERROR-NORMAL
			; 5 - 1: FORCE NOT(RTS) LOW
			; 6 - 0:
			; 7 - 0: NORMAL OPERATION
URT025	DEFB	01001110B	; PROGRAM USART MODE 1 REGISTER
			; BIT 0 - 0:
			; 1 - 1: ASYNCHRONOUS 16X
			; 2 - 1:
			; 3 - 1: 8 BITS PER CHARACTER
			; 4 - 0: DISABLE PARITY
			; 5 - 0: PARITY (ODD OR EVEN) -DISABLED
			; 6 - 1:

URT026	DEFB	00000000B	7 - 0: 1 STOP BIT
			; PROGRAM USART MODE 2 REGISTER
			; BIT 0 - 0:
			; 1 - 0:
			; 2 - 0:
			; 3 - 0: BAUD RATE SELECTOR-NULL CODE
			; 4 - 0: EXTERNAL RECEIVER CLOCK
			; 5 - 0: EXTERNAL TRANSMITTER CLOCK
			; 6 - 0: NOT USED
			; 7 - 0: NOT USED
			; ASSOC CTC - CTC1, CH2 ADDRESS
URT027	DEFB	12H	; CTC MODE COMMAND
URT028	DEFB	01000101B	; BIT 0 - 1: CONTROL WORD
			; 1 - 0: RESET-CONTINUED OPERATION
			; 2 - 1: TIME CONSTANT NEXT WORD
			; 3 - 0: AUTOMATIC TRIGGER
			; 4 - 0: SELECT (-)EDGE
			; 5 - 0: PRESCALER OF 16
			; 6 - 1: COUNTER MODE
			; 7 - 0: DISABLE INTERRUPTS
URT029	DEFB	04H	; CTC TIME CNST PRESCALER FOR 19.2 KBAUD
URT03L			; USART 3 INITIALIZATION DATA LIST
URT030	DEFB	08H	; DATA ADDRESS
URT031	DEFB	09H	; STATUS ADDRESS
URT032	DEFB	0AH	; MODE ADDRESS
URT033	DEFB	0BH	; COMMAND ADDRESS
URT034	DEFB	00100110B	; PROGRAM USART COMMAND REGISTER
			; BIT 0 - 0: DISABLE TRANSMIT
			; 1 - 1: FORCE NOT(DTR) LOW
			; 2 - 1: DISABLE RECEIVE
			; 3 - 0: ASYNCH-NORMAL
			; 4 - 0: RESET ERROR-NORMAL
			; 5 - 1: FORCE NOT(RTS) LOW
			; 6 - 0:

URT035	DEFB	01001110B	7 - 0: NORMAL OPERATION
			; PROGRAM USART MODE 1 REGISTER
			; BIT 0 - 0:
			; 1 - 1: ASYNCHRONOUS 16X
			; 2 - 1:
			; 3 - 1: 8 BITS PER CHARACTER
			; 4 - 0: DISABLE PARITY
			; 5 - 0: PARITY (ODD OR EVEN)-DISABLED
			; 6 - 1:
			; 7 - 0: 1 STOP BIT
URT036	DEFB	00000000B	; PROGRAM USART MODE 2 REGISTER
			; BIT 0 - 0:
			; 1 - 0:
			; 2 - 0:
			; 3 - 0: BAUD RATE SELECTOR-NULL CODE
			; 4 - 0: EXTERNAL RECEIVER CLOCK
			; 5 - 0: EXTERNAL TRANSMITTER CLOCK
			; 6 - 0: NOT USED
			; 7 - 0: NOT USED
URT037	DEFB	15H	; ASSOC CTC - CTC2, CH1 ADDRESS
URT038	DEFB	01000101B	; CTC MODE COMMAND
			; BIT 0 - 1: CONTROL WORD
			; 1 - 0: RESET-CONTINUED OPERATION
			; 2 - 1: TIME CONSTANT NEXT WORD
			; 3 - 0: AUTOMATIC TRIGGER
			; 4 - 0: SELECT (-)EDGE
			; 5 - 0: PRESCALER OF 16
			; 6 - 1: COUNTER MODE
			; 7 - 0: DISABLE INTERRUPTS
URT039	DEFB	04H	; CTC TIME CNST PRESCALER FOR 19.2 KBAUD
URT040L			; USART 4 INITIALIZATION DATA LIST
URT040	DEFB	0CH	; DATA ADDRESS
URT041	DEFB	0DH	; STATUS ADDRESS
URT042	DEFB	0EH	; MODE ADDRESS

URT043	DEFB	0FH	COMMAND ADDRESS
URT044	DEFB	00100110B	PROGRAM USART COMMAND REGISTER
			BIT 0 - 0: DISABLE TRANSMIT
			1 - 1: FORCE NOT(DTR) LOW
			2 - 1: DISABLE RECEIVE
			3 - 0: ASYNCH-NORMAL
			4 - 0: RESET ERROR-NORMAL
			5 - 1: FORCE NOT(RTS) LOW
			6 - 0:
			7 - 0: NORMAL OPERATION
URT045	DEFB	01001110B	PROGRAM USART MODE 1 REGISTER
			BIT 0 - 0:
			1 - 1: ASYNCHRONOUS 16X
			2 - 1:
			3 - 1: 8 BITS PER CHARACTER
			4 - 0: DISABLE PARITY
			5 - 0: PARITY (ODD OR EVEN)-DISABLED
			6 - 1:
			7 - 0: 1 STOP BIT
URT046	DEFB	00000000B	PROGRAM USART MODE 2 REGISTER
			BIT 0 - 0:
			1 - 0:
			2 - 0:
			3 - 0: BAUD RATE SELECTOR-NULL CODE
			4 - 0: EXTERNAL RECEIVER CLOCK
			5 - 0: EXTERNAL TRANSMITTER CLOCK
			6 - 0: NOT USED
			7 - 0: NOT USED
URT047	DEFB	16H	ASSOC CTC - CTC2, CH2 ADDRESS
URT048	DEFB	01000101B	CTC MODE COMMAND
			BIT 0 - 1: CONTROL WORD
			1 - 0: RESET-CONTINUED OPERATION
			2 - 1: TIME CONSTANT NEXT WORD
			3 - 0: AUTOMATIC TRIGGER
			4 - 0: SELECT (-)EDGE

```

;      5 - 0: PRESCALER OF 16
;      6 - 1: COUNTER MODE
;      7 - 0: DISABLE INTERRUPTS
;      ; CTC TIME CNST PRESCALER FOR 19.2 KBAUD
URT049  DEFB  04H

```

```

;*****

```

```

*EJECT

```

```

;*****
;*****
;*****

```

```

;PROCEDURE      TRNMIT      TRANSMIT PROCEDURE

```

```

;

```

```

;      THE PURPOSE OF THIS PROCEDURE IS TO ENABLE A
;      TRANSMIT INTERRUPT FROM A PLZ MODULE.
;

```

```

;INPUT -      THIS PROCEDURE EXPECTS ONE INPUT PARAMETER: THE
;      DATA PORT ADDRESS OF THE USART SUPPORTING THE CHANNEL
;      OVER WHICH THE DATA IS TO BE OUTPUT. THIS ADDRESS IS
;      TO BE IN GLOBAL PARAMETER TPRADD.
;

```

```

;PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE IX REG FOR
;      NORMALIZATION AT THE RETURN.

```

```

;      THE NEXT SECTION CONVERTS THE DATA PORT ADDRESS
;      TO THE COMMAND PORT ADDRESS AND LOADS IT INTO THE C REG.
;      THE TRANSMIT ENABLE BIT IS SET IN THE RETRIEVED COMMAND
;      WORD, AND SENT BACK OUT TO THE USART. THIS ACTION
;      CAUSES THE ACTUAL INTERRUPT.

```

```

;      FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
;      RECOVERED, AND A RETURN EXECUTED.
;

```

```

;OUTPUT -      A COMMAND WORD IS OUTPUT TO THE APPROPRIATE
;      (AS IDENTIFIED BY THE INPUT PARAMETER) USART WHICH
;      CONTAINS A SET TRANSMIT ENABLE BIT.
;

```

```

;INTERFACE -      THE TRANSMIT INTERRUPT IS ENABLED THROUGH A
;      PRIORITY INTERRUPT CONTROLLER (PIC). WHEN AN INTERRUPT

```



```
POP IX      ; RESTORE IX
POP HL      ; RECOVER RETURN ADDRESS
JP (HL)     ; RETURN
```

* * * * *

EJECT

```
URTR01
;PROCEDURE
I/O RECEIVE INTERRUPT CONTROLLER
```

THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
CHANNEL 1 RECEIVE INTERRUPTS.

```

; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; ; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. LCOLNS IS
; ; THE NEXT SERVICE POSITION; LCOLNE IS THE NEXT EMPTY POSITION;
; ; AND LCOLSZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
; ; LCOLTB.

```

```

;PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
;PROGRAM'S REGISTERS. THE BYTE IS LOADED INTO
;THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;FOR WRAPAROUND IF NECESSARY. FINALLY, THE INTERRUPTED
;PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;

```

```

;OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE LC01TB AND
;        THE LC01NE POSITION IS UPDATED TO REFLECT THE BYTE
;        INSERTION.

```

```

;INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE PIC. AS AN INTERRUPT IS IDENTIFIED, THE PIC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE

```

```

;
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
; ***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
URTR01:

EXTERNAL LC01TB LC01NE LC01SZ

EX AF,AF' ; SAVE REGS OF INTERRUPTED PROGRAM
EXX
PUSH IX
PUSH IY

IN A,(0) ; INPUT THE BYTE

LD DE,LC01TB ; SET HL TO NEXT EMPTY BUFF LOCATION
LD HL,(LC01NE)
ADD HL,DE

LD (HL),A ; LD BYTE INTO EMPTY BUFF LOCATION

LD HL,(LC01NE) ; LD EMPTY LOC POINTER AND
INC HL ; INC EMPTY LOC POINTER
LD (LC01NE),HL
LD DE,(LC01SZ) ; LD BUFF SIZE FOR CHECK
SBC HL,DE ; IF AT AND OF BUFF, RESET TO LOC ZERO
JR NZ,URLJ10
LD HL,0
LD (LC01NE),HL

LD A,(PICCMD) ; LD PIC COMMAND
OUT (PICADD),A ; SEND COMMAND

URLJ10

```

```

EX AF,AF'      ; RESTORE CALLING PROG'S REGS
EXX
POP IY
POP IX

EI             ; ENABLE INTERRUPTS

RETI          ; RETURN

*****
;EJECT
*****
;PROCEDURE      URTR02      I/O RECEIVE INTERRUPT CONTROLLER
;
;      THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;      CHANNEL 2 RECEIVE INTERRUPTS.
;
;      THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
;      TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. LC02NS IS
;      THE NEXT SERVICE POSITION; LC02NE IS THE NEXT EMPTY POSITION;
;      AND LC02SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;      LC02TB.
;
;PROCESSING -   THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
;               PROGRAM'S REGISTERS. THE BYTE IS LOADED INTO
;               THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
;               FOR WRAPAROUND IF NECESSARY. FINALLY, THE INTERRUPTED
;               PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
;               INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -      THE BYTE RECEIVED IS LOADED INTO THE LC02TB AND
;               THE LC02NE POSITION IS UPDATED TO REFLECT THE BYTE
;               INSERTION.
;
*****

```

```

;INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE PIC. AS AN INTERRUPT IS IDENTIFIED, THE PIC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES - NONE.
;*****
;
;***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS

```

URTR02:

EXTERNAL LC02TB LC02NE LC02SZ

```

EX AF,AF' ; SAVE REGS OF INTERRUPTED PROGRAM
EXX
PUSH IX
PUSH IY

IN A,(4) ; INPUT THE BYTE

LD DE,LC02TB ; SET HL TO NEXT EMPTY BUFF LOCATION
LD HL,(LC02NE)
ADD HL,DE

LD (HL),A ; LD BYTE INTO EMPTY BUFF LOCATION

LD HL,(LC02NE) ; LD EMPTY LOC POINTER AND
INC HL ; INC EMPTY LOC POINTER
LD (LC02NE),HL
LD DE,(LC02SZ) ; LD BUFF SIZE FOR CHECK
SBC HL,DE ; IF AT AND OF BUFF, RESET TO LOC ZERO
JR NZ,UR2JJ10
LD HL,0
LD (LC02NE),HL

```

UR2JJ10

```

LD A,(PICCMD) ; LD PIC COMMAND
OUT (PICADD),A ; SEND COMMAND

EX AF,AF' ; RESTORE CALLING PROG'S REGS
EXX
POP IY
POP IX

EI ; ENABLE INTERRUPTS

RETI ; RETURN

```

```

;*****
;EJECT
;*****
;*****
;PROCEDURE URTR03 I/O RECEIVE INTERRUPT CONTROLLER
;
; THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
; CHANNEL 3 RECEIVE INTERRUPTS.
;
; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. LC03NS IS
; THE NEXT SERVICE POSITION; LC03NE IS THE NEXT EMPTY POSITION;
; AND LC03SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
; LC03TB.
;
; PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
; PROGRAM'S REGISTERS. THE BYTE IS LOADED INTO
; THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
; FOR WRAPAROUND IF NECESSARY. FINALLY, THE INTERRUPTED
; PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,
; INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
; OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE LC03TB AND

```

```

;
;
;
; THE LC03NE POSITION IS UPDATED TO REFLECT THE BYTE
; INSERTION.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE PIC. AS AN INTERRUPT IS IDENTIFIED, THE PIC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
;
; ***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
URTR03:

```

```

EXTERNAL LC03TB LC03NE LC03SZ

EX AF,AF'      ; SAVE REGS OF INTERRUPTED PROGRAM
EXX
PUSH IX
PUSH IY

IN A,(8)      ; INPUT THE BYTE

LD DE,LC03TB  ; SET HL TO NEXT EMPTY BUFF LOCATION
LD HL,(LC03NE)
ADD HL,DE

LD (HL),A     ; LD BYTE INTO EMPTY BUFF LOCATION

LD HL,(LC03NE) ; LD EMPTY LOC POINTER AND
INC HL        ; INC EMPTY LOC POINTER
LD (LC03NE),HL
LD DE,(LC03SZ) ; LD BUFF SIZE FOR CHECK
SBC HL,DE     ; IF AT AND OF BUFF, RESET TO LOC ZERO
JR NZ,UR3JJ10

```

```

LD HL,0
LD (LC03NE),HL

LD A,(PICCMD) ; LD PIC COMMAND
OUT (PICADD),A ; SEND COMMAND

EX AF,AF' ; RESTORE CALLING PROG'S REGS
EXX
POP IY
POP IX

EI ; ENABLE INTERRUPTS

RETI ; RETURN

```

UR3J10

```

;*****
;EJECT
;*****
;*****
;PROCEDURE URTR04 I/O RECEIVE INTERRUPT CONTROLLER
;
; THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
; CHANNEL 4 RECEIVE INTERRUPTS.
;
; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. LC04NS IS
; THE NEXT SERVICE POSITION; LC04NE IS THE NEXT EMPTY POSITION;
; AND LC04SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
; LC04TB.
;
; PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
; PROGRAM'S REGISTERS. THE BYTE IS LOADED INTO
; THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
; FOR WRAPAROUND IF NECESSARY. FINALLY, THE INTERRUPTED
; PROGRAM'S REGISTERS ARE RESTORED, THE PIC RE-INITIALIZED,

```

```

; INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
; OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE LC04TB AND
; THE LC04NE POSITION IS UPDATED TO REFLECT THE BYTE
; INSERTION.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE PIC. AS AN INTERRUPT IS IDENTIFIED, THE PIC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; ***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
URTR04:

```

```

EXTERNAL LC04TB LC04NE LC04SZ
EX AF,AF' ; SAVE REGS OF INTERRUPTED PROGRAM
EXX
PUSH IX
PUSH IY

IN A,(0CH) ; INPUT THE BYTE

LD DE,LC04TB ; SET HL TO NEXT EMPTY BUFF LOCATION
LD HL,(LC04NE)
ADD HL,DE

LD (HL),A ; LD BYTE INTO EMPTY BUFF LOCATION

LD HL,(LC04NE) ; LD EMPTY LOC POINTER AND
INC HL ; INC EMPTY LOC POINTER
LD (LC04NE),HL

```



```

; BACK TO THE DATA PORT AND THE BYTE IS OUTPUT. THE PROC
; WAITS AGAIN FOR A READY CONDITION AND THEN RESETS THE
; TRANSMIT ENABLE CONDITION TO GET OUT OF INTERRUPT. FINALLY,
; THE PRIORITY INTERRUPT CONTROLLER (PIC) IS RE-INITIALIZED,
; AND A RETURN FROM INTERRUPT IS EXECUTED.
;
; OUTPUT - A BYTE OF DATA IS OUTPUT ON THE DATA LINE, AND THE
; PIC IS RESET TO ENABLE INTERRUPTS.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE PIC. AS AN INTERRUPT IS IDENTIFIED, THE PIC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
; ***** ; PROCEDURE TO HANDLE TRANSMIT I/O INTERRUPTS
URTRN:

```

EXTERNAL TDAADD TPRADD

```

LD A,(TPRADD) ; LD PORT
LD C,+1 ; CNVRT TO STATUS PORT ADD
ADD A,C
LD C,A

```

```

UTRNL1 IN A,(C) ; WAIT UNTIL READY TO TRANSMIT
BIT 0,A
JR Z,UTRNL1

```

```

LD A,-1 ; CNVRT TO DATA PORT ADD
ADD A,C
LD C,A

```

```

LD DE,(TDAADD) ; LD DATA
LD A,(DE)

OUT (C),A ; OUTPUT A BYTE

LD A,+1 ; CNVRT TO STATUS PORT ADD
ADD A,C
LD C,A
IN A,(C) ; WAIT UNTIL READY TO TRANSMIT
BIT 0,A
JR Z,UTRNL2

LD A,+2 ; CNVRT TO COMMAND PORT ADD
ADD A,C
LD C,A
IN A,(C) ; RESET TRANSMIT ENABLE
RES 0,A
OUT (C),A

LD A,(PICCMD) ; RESET PIC
OUT (PICADD),A

EI ; ENABLE INTERRUPTS
RETI ; RETURN

```

UTRNL2

Appendix F Section IV

This section of Appendix F contains the software listings which comprise the Network Operating System of UNID#2 (UNID#1 is the same except for THIS_UNID_NBR and minor textual differences).


```

; INPUT - THE ADDRESS FOR RETURNING TO THE CALLING
; PROCEDURE IS LOCATED ON THE TOP OF THE STACK.
;
; PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE IX REG FOR
; NORMALIZATION AT THE RETURN. THE SIO AND ASSOCIATED CTC ARE
; THEN INITIALIZED. THIS INITIALIZATION IS ACCOMPLISHED
; BY USING TWO DATA LISTS CONTAINING THE INITIALIZATION
; COMMANDS AND PORT ADDRESSES FOR THE SIO AND CTC.
;
; INTERRUPT REGISTER (I) INITIALIZATION IS NEXT
; WITH THE I REG BEING LOADED WITH THE ADDRESS OF THE I/O
; VECTOR TABLE (IOVCTB). THIS TABLE MUST BE LOCATED ON AN
; EVEN 100 HEX MEMORY BOUNDARY (1600, 1700, ETC.). WHEN AN
; INTERRUPT IS IDENTIFIED BY THE SIO, THE I REG SUPPLIES
; THE HIGH 8 BITS AND THE SIO SUPPLIES THE LOW 8 BITS OF THE
; ADDRESS TO THE I/O CONTROLLER THAT WILL SERVICE THE INTERRUPT.
; WITH THE IOVCTB TABLE ON AN EVEN MEMORY BOUNDARY, ONLY THE
; 8 HIGH BITS ARE REQUIRED TO BE LOADED AS THE LOWER BITS
; ARE ALL ZEROS.
;
; THE 280 INTERRUPT MODE IS SET TO 2 AND INTERRUPTS
; ENABLED. AT THIS POINT, THE UNID IS CONFIGURED FOR
; INTERRUPT DRIVEN COMMUNICATIONS ON THE NETWORK SIDE FOR
; BOTH CHANNEL-A AND CHANNEL-B.
;
; FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
; RECOVERED, AND A RETURN EXECUTED.
;
; TWO PROCEDURE STUBS, ONE FOR SPECIAL RECEIVE ACTION
; ON CHANNEL-A AND THE OTHER FOR CHANNEL-B, ARE INCLUDED FOR FUTURE
; EXPANSION.
;
; OUTPUT - THE SIO AND CTC ARE PASSED INITIALIZATION COMMANDS.
; ADDITIONALLY, THE I REG IS SET WITH THE HIGH
; 8 BITS OF IOVCTB ADDRESS, AND INTERRUPTS ENABLED IN MODE TWO.
;
; INTERFACE - THIS PROCEDURE IS CALLED FROM N.MAIN_U2, THE MAIN LINE
; DRIVER OF THE UNID NETWORK OPERATING SYSTEM.

```

```

;
; THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
; COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
; PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
; AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
; OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
; PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
; NOTES - NONE.
;
; *****
;
; ***** NOTE *****
;
; THIS MODULE IS ORG'D IN TWO AREAS:
; THE CODE, AND THE TABLE IOVCTB.
; IMPORTANT INFO ABOUT THIS ORG IS
; IN THE PROC DOCUMENTATION ABOVE.
; *****
;
; IO VECTOR ADDRESS TABLE
; SIO B CHANNEL
; NUMBER OF ITEMS IN LIST
; SIO B RECEIVE CALLS RECCHB
; SIO B TRANSMIT CALLS XMTCHB (STUB, NOT USED)
; SIO A CHANNEL
; NUMBER OF ITEMS IN LIST
; SIO A RECEIVE CALLS RECCHA
; SIO A TRANSMIT CALLS XMTCHA (STUB, NOT USED)
;
; CTC INTERRUPT VECTOR TABLE
; CTC INTERRUPT FOR CHANNEL-B
; CTC TIME OUT CALLS TOUTCB
; CTC INTERRUPT FOR CHANNEL-A
; CTC TIME OUT CALLS TOUTCA
;
; ***** NOTE *****
; THIS MODULE IS ORG'D IN TWO AREAS:

```

```

; THE CODE, AND THE TABLE IOVCTB.
; IMPORTANT INFO ABOUT THIS ORG IS
; IN THE PROCEDURE DOCUMENTATION ABOVE.
; **** NOTE ****

```

```

EXTERNAL TIMCHA, TIMCHB, CTCNOA, CTCNOB, MAXNOA, MAXNOB
; ABOVE EXTERNALS COME FROM N.MAIN_U2

```

INSIO:

```

; PROCEDURE TO INITIALIZE SIO COMMUNICATIONS
; STORE IX FOR RETURN

```

```

LD HL, SIOCHA
LD C, (HL)
; LD ADD OF SIO CHANNEL-A PARAMETER LIST
; LD ADD OF SIO PORT A CMD/STATUS

```

```

INC HL
LD B, (HL)
; INC TO NEXT BYTE IN LIST
; LD NBR OF ENTRIES IN PORT A LIST

```

```

INC HL
OTIR
; INC TO NEXT BYTE IN LIST
; OUTPUT REMAINING BYTES TO SIO

```

```

LD HL, SIOCHB
LD C, (HL)
; LD ADD OF SIO CHANNEL-B PARAMETER LIST
; LD ADD OF SIO PORT B CMD/STATUS

```

```

INC HL
LD B, (HL)
; INC TO NEXT BYTE IN LIST
; LD NBR OF ENTRIES IN PORT B LIST

```

```

INC HL
OTIR
; INC TO NEXT BYTE IN LIST
; OUTPUT REMAINING BYTES TO SIO

```

```

; INITIALIZE CTC
LD HL, CTCLST
LD C, (HL)
; LD ADD OF CTC PARAMETER LIST
; LD ADD OF CTC CHANNEL 0

```

```

INC HL
LD E, (HL)
; INC TO NEXT BYTE IN LIST
; LD LOW BYTE OF INTERRUPT ADD

```



```

OUT (C),E      ; OUTPUT VECTOR ADD TO CTC CH 0
INC HL
INC HL      ; INC TWO BYTES IN LIST
OUTI
OUTI      ; SET OPERATING MODE
          ; SET TIME CONSTANT
INC C
OUTI
OUTI      ; INC C TO CTC CH 1 PORT ADD
          ; SET OPERATING MODE
          ; SET TIME CONSTANT
INC C
OUTI
OUTI      ; INC C TO CTC CH 2 PORT ADD
          ; SET OPERATING MODE
          ; SET TIME CONSTANT
INC C
OUTI
OUTI      ; INC C TO CTC CH 3 PORT ADD
          ; SET OPERATING MODE
          ; SET TIME CONSTANT
LD HL,IOVCTB  ; LD ADD OF VECTOR ADDRESS TABLE
LD A,H        ; LD HIGH 8 BITS OF ADD INTO I REG FOR BASE
LD I,A
IM 2
EI      ; SET INTERRUPT MODE TO VECTOR ADD MODE
          ; ENABLE INTERRUPTS
POP IX
POP HL      ; RESTORE IX
          ; RECOVER RETURN ADDRESS
JP (HL)      ; RETURN
NOP
RET      ; SIO CHANNEL-A XMIT STUB. NOT CURRENTLY USED.

```

XMTCHA

XMTCHB	NOP	; SIO CHANNEL-B XMIT STUB. NOT CURRENTLY USED.
	RET	; RETURN
TOUTCA:	PUSH IX	; TIME OUT PROCEDURE FOR CHANNEL-A
	LD HL,CTCNOA	; SAVE IX REGISTERS
	INC (HL)	; PLACE ADDRESS OF CTCNOA IN HL
	POP IX	; INCREMENT CTC COUNTER BY '1'
	POP HL	; RESTORE IX REGISTERS
	JP (HL)	; GET RETURN ADDRESS IN HL
		; RETURN TO CALLING PROGRAM
TOUTCB:	PUSH IX	; TIME OUT PROCEDURE FOR CHANNEL-B
	LD HL,CTCNOB	; SAVE IX REGISTERS
	INC (HL)	; PLACE ADDRESS OF CTCNOB IN HL
	POP IX	; INCREMENT CTC COUNTER BY '1'
	POP HL	; RESTORE IX REGISTERS
	JP (HL)	; GET RETURN ADDRESS IN HL
		; RETURN TO CALLING PROGRAM
A_DATA	EQU 00H	; SIO PORT A DATA ADDRESS
B_DATA	EQU 01H	; SIO PORT B DATA ADDRESS
A_STAT	EQU 02H	; SIO PORT A STATUS/COMMAND ADDRESS
B_STAT	EQU 03H	; SIO PORT B STATUS/COMMAND ADDRESS
CTC_0	EQU 04H	; CTC CHANNEL 0 ADDRESS
CTC_1	EQU 05H	; CTC CHANNEL 1 ADDRESS
CTC_2	EQU 06H	; CTC CHANNEL 2 ADDRESS
CTC_3	EQU 07H	; CTC CHANNEL 3 ADDRESS
TC1	EQU 00H	; TIME CONSTANT = 256 FOR CHANNEL-1
TC2	EQU 00H	; TIME CONSTANT = 256 FOR CHANNEL-2
TC3	EQU 00H	; TIME CONSTANT = 256 FOR CHANNEL-3
FALSE	EQU 00H	; FOR TRUE OR FALSE FLAG
		;DEFINES

CTCLST	DEFB	CTC_0	; CTC CHANNEL-0 ADDRESS
	DEFW	CTCVEC	; CTC INTERRUPT VECTOR ADDRESS
	DEFB	01000111B	; CHANNEL-0
			; BIT 0 - 1: CHANNEL CONTROL WORD
			; 1 - 1: RESET CHANNEL
			; 2 - 1: TIME CONSTANT NEXT WORD
			; 3 - 0: NA (TIMER MODE ONLY)
			; 4 - 0: (-)EDGE DEC COUNTER
			; 5 - 0: NA (TIMER MODE ONLY)
			; 6 - 1: COUNTER MODE SELECTED
			; 7 - 0: CHANNEL INT DISABLED
			; TIME CONSTANT FOR CHANNEL-0
	DEFB	00000001B	; BIT 0 - 1:
			; 1 - 0:
			; 2 - 0:
			; 3 - 0:
			; 4 - 0:
			; 5 - 0:
			; 6 - 0:
			; 7 - 0:
			; CHANNEL-1
	DEFB	01000111B	; BIT 0 - 1: CHANNEL CONTROL WORD
			; 1 - 1: RESET CHANNEL
			; 2 - 1: TIME CONSTANT NEXT WORD
			; 3 - 0: NA (TIMER MODE ONLY)
			; 4 - 0: (-)EDGE DEC COUNTER
			; 5 - 0: NA (TIMER MODE ONLY)
			; 6 - 1: COUNTER MODE SELECTED
			; 7 - 0: CHANNEL INT DISABLED

DEFB	10000000B	; TIME CONSTANT-1 (9600 BAUD)
		; BIT 0 - 0:
		; 1 - 0:
		; 2 - 0:
		; 3 - 0:
		; 4 - 0:
		; 5 - 0:
		; 6 - 0:
		; 7 - 1:
DEFB	01000111B	; CHANNEL-2
		; BIT 0 - 1: CHANNEL CONTROL WORD
		; 1 - 1: RESET CHANNEL
		; 2 - 1: TIME CONSTANT NEXT WORD
		; 3 - 0: NA (TIMER MODE ONLY)
		; 4 - 0: (-)EDGE DEC COUNTER
		; 5 - 0: NA (TIMER MODE ONLY)
		; 6 - 1: COUNTER MODE SELECTED
		; 7 - 0: CHANNEL INT DISABLED
DEFB	00000001B	; TIME CONSTANT-2
		; BIT 0 - 1:
		; 1 - 0:
		; 2 - 0:
		; 3 - 0:
		; 4 - 0:
		; 5 - 0:
		; 6 - 0:
		; 7 - 0:
DEFB	01000111B	; CHANNEL-3
		; BIT 0 - 1: CHANNEL CONTROL WORD
		; 1 - 1: RESET CHANNEL
		; 2 - 1: TIME CONSTANT NEXT WORD

```

; 3 - 0: NA (TIMER MODE ONLY)
; 4 - 0: (-)EDGE DEC COUNTER
; 5 - 0: NA (TIMER MODE ONLY)
; 6 - 1: COUNTER MODE SELECTED
; 7 - 0: CHANNEL INT DISABLED

```

```

; TIME CONSTANT-3

```

```

DEFB 00000001B

```

```

; BIT 0 - 1:
; 1 - 0:
; 2 - 0:
; 3 - 0:
; 4 - 0:
; 5 - 0:
; 6 - 0:
; 7 - 0:

```

```

; PORT A
; SIO PORT A CMD/STAT ADD
; NUMBER OF ENTRIES IN LIST

```

```

SIOCHA DEFB A_STAT
        DEFB 16D

```

```

DEFB 00000100B
; SELECT WR4
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-4
; 2 - 1: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

```

```

DEFB 00100000B
; PROGRAM WRITE REGISTER-4
; BIT 0 - 0: PARITY-NOT USED
; 1 - 0: PARITY(EVEN-ODD) -NOT USED
; 2 - 0: |
; 3 - 0: | SYNC MODES

```

```

; 4 - 0: |
; 5 - 1: | SDL C MODE
; 6 - 0: |
; 7 - 0: | X1 CLOCK RATE

```

DEFB 00000000B

```

; SELECT WR0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

```

DEFB 01010000B

```

; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: |
; 4 - 1: | RESET EXT/STAT INT
; 5 - 0: |
; 6 - 1: |
; 7 - 0: | RESET CRC CHECKER

```

DEFB 00000001B

```

; SELECT WR1
; BIT 0 - 1: |
; 1 - 0: | WRITE REGISTER-1
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

```

```

DEFB      00011000B

; PROGRAM WRITE REGISTER-1
; BIT 0 - 0: EXT/STAT INT DISABLE
;      1 - 0: XMIT INT DISABLE
;      2 - 0: STAT AFFECT REG-CH-B ONLY
;      3 - 1: |
;      4 - 1: | INT ON ALL RX CHAR
;      5 - 0: WAIT/READY TO XMMIT BUFFER
;      6 - 0: WAIT FUNCTION
;      7 - 0: WAIT-READY DISABLE

```

```

DEFB      00000011B

; SELECT WR3
; BIT 0 - 1: |
;      1 - 1: | WRITE REGISTER-3
;      2 - 0: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE

```

```

DEFB      11001001B

; PROGRAM WRITE REGISTER-3
; BIT 0 - 1: RECEIVE ENABLE
;      1 - 0: SYNC CHAR LOAD INHIBIT
;      2 - 0: ADD SEARCH MODE DISABLED
;      3 - 1: RECEIVER CRC ENABLED
;      4 - 0: HUNT PHASE DISABLED
;      5 - 0: NOT(DCD), NOT(CTS)
;      6 - 1: | ARE INPUTS TO RR0
;      7 - 1: | RECEIVE 8-BITS PER CHAR

```

```

DEFB      00000000B

; SELECT WR0
; BIT 0 - 0: |
;      1 - 0: | WRITE REGISTER-0
;      2 - 0: |

```

```

;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE
;
; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
;      1 - 0: | WRITE REGISTER-0
;      2 - 0: |
;      3 - 0: |
;      4 - 1: | RESET EXT-STAT INT
;      5 - 0: |
;      6 - 0: |
;      7 - 1: | RESET TX GEN

```

DEFB 10010000B

```

; SELECT WR5
; BIT 0 - 1: |
;      1 - 0: | WRITE REGISTER-5
;      2 - 1: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE

```

DEFB 00000101B

```

; PROGRAM WRITE REGISTER-5
; BIT 0 - 0: TRANSMIT CRC ENABLE
;      1 - 1: SET RTS TO HIGH
;      2 - 0: (X16+X12+X5+1) POLYNOMIAL
;      3 - 1: TRANSMIT ENABLE
;      4 - 0: TXD RETURNS TO MARKING
;      5 - 1: |
;      6 - 1: | TRANSMIT 8-BITS PER CHAR
;      7 - 1: SET DTR HIGH

```

DEFB 11101010B

DEFB 00000110B

```
; SELECT WR6
; BIT 0 - 0: |
; 1 - 1: | WRITE REGISTER-6
; 2 - 1: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE
;
```

DEFB 00000000B

```
; REG-6 MIT SYNC CHAR FOR MONOSYNC ONLY
; BIT 0 - 0:
; 1 - 0:
; 2 - 0:
; 3 - 0:
; 4 - 0:
; 5 - 0:
; 6 - 0:
; 7 - 0:
;
```

DEFB 00000111B

```
; SELECT WR7
; BIT 0 - 1: |
; 1 - 1: | WRITE REGISTER-7
; 2 - 1: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE
;
```

DEFB 01111110B

```
; PROGRAM REGISTER-7 (SLDC FLAG)
; BIT 0 - 0: SYNC 8
; 1 - 1: SYNC 9
; 2 - 1: SYNC 10
;
```

```

;      3 - 1: SYNC 11
;      4 - 1: SYNC 12
;      5 - 1: SYNC 13
;      6 - 1: SYNC 14
;      7 - 0: SYNC 15

; PORT B
; SIO PORT B CMD/STAT ADD
; NBR OF ENTRIES IN LIST

; SELECT WR4
; BIT 0 - 0: |
;      1 - 0: | WRITE REGISTER-4
;      2 - 0: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE

; PROGRAM WRITE REGISTER-4
; BIT 0 - 0: PARITY-NOT USED
;      1 - 0: PARITY (EVEN-ODD)-NOT USED
;      2 - 0: |
;      3 - 0: | SYNC MODES
;      4 - 0: |
;      5 - 1: | SDLC MODE
;      6 - 0: |
;      7 - 0: | X1 CLOCK RATE

; SELECT WR0
; BIT 0 - 0: |
;      1 - 0: | WRITE REGISTER-0
;      2 - 0: |
;      3 - 0: NULL CODE

```

SIOCHB	DEFB	B_STAT
	DEFB	18D
	DEFB	00000100B
	DEFB	001000000B
	DEFB	000000000B

```

DEFB 01010000B
;
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE
;
; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: |
; 4 - 1: | RESET EXT-STAT INT
; 5 - 0: |
; 6 - 1: |
; 7 - 0: | RESET RX CRC CHECKER
;

DEFB 00000000B
; SELECT WR1
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: | NULL CODE
; 4 - 0: | NULL CODE
; 5 - 0: | NULL CODE
; 6 - 0: | NULL CODE
; 7 - 0: | NULL CODE
;

DEFB 00011000B
; PROGRAM WRITE REGISTER-1
; BIT 0 - 0: EXT-STAT INT DISABLE
; 1 - 0: TRANSMITTER INT DISABLE
; 2 - 0: STAT AFFECT VECT-CH-B ONLY
; 3 - 1: |
; 4 - 1: | INT ON ALL RX CHAR
; 5 - 0: WAIT-READY ON XMIT BUFFER
; 6 - 0: WAIT FUNCTION
; 7 - 0: WAIT-READY DISABLE
;

```

```

DEFB 00000010B
; SELECT WR2
; BIT 0 - 0: |
; 1 - 1: | WRITE REGISTER-2
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

DEFB 00000000B
; WR-2 IS VECTOR INTERRUPT REGISTER
; BIT 0 - 0:
; 1 - 0:
; 2 - 0:
; 3 - 0:
; 4 - 0:
; 5 - 0:
; 6 - 0:
; 7 - 0:

DEFB 00000011B
; SELECT WR3
; BIT 0 - 1: |
; 1 - 1: | WRITE REGISTER-3
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

DEFB 11001001B
; PROGRAM WRITE REGISTER-3
; BIT 0 - 1: RECEIVE ENABLE
; 1 - 0: SYNC CHAR LOAD INHIBIT
; 2 - 0: ADD SEARCH MODE DISABLED
; 3 - 1: RECEIVER CRC ENABLED

```

```

DEFB 00000000B
;
; 4 - 0: HUNT PHASE DISABLED
; 5 - 0: NOT(DCD), NOT(CTS)
; ARE INPUTS TO RR0
; 6 - 1: |
; 7 - 1: | RECEIVE 8 BITS PER CHAR
;
; SELECT WR0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE
;

DEFB 10010000B
; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: |
; 4 - 1: | RESET EXT-STAT INT
; 5 - 0: |
; 6 - 0: |
; 7 - 1: | RESET TX GEN
;

DEFB 00000101B
; SELECT WR5
; BIT 0 - 1: |
; 1 - 0: | WRITE REGISTER-5
; 2 - 1: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE
;

```

DEFB	11101010B	<pre> ; PROGRAM WRITE REGISTER-5 ; BIT 0 - 0: TRANSMIT CRC ENABLE ; 1 - 1: SET RTS HIGH ; 2 - 0: (X16+X12+X5+1) POLYNOMIAL ; 3 - 1: TRANSMIT ENABLE ; 4 - 0: TxD RETURNS TO MARKING ; 5 - 1: ; 6 - 1: TRANSMIT 8 BITS PER CHAR ; 7 - 1: SET DTR HIGH </pre>
DEFB	00000110B	<pre> ; SELECT WR6 ; BIT 0 - 0: ; 1 - 1: WRITE REGISTER-6 ; 2 - 1: ; 3 - 0: NULL CODE ; 4 - 0: NULL CODE ; 5 - 0: NULL CODE ; 6 - 0: NULL CODE ; 7 - 0: NULL CODE </pre>
DEFB	00000000B	<pre> ; REG 6 XMIT SYNC CHAR, MONOSYNC ONLY ; BIT 0 - 0: ; 1 - 0: ; 2 - 0: ; 3 - 0: ; 4 - 0: ; 5 - 0: ; 6 - 0: ; 7 - 0: </pre>
DEFB	00000111B	<pre> ; SELECT WR7 ; BIT 0 - 1: ; 1 - 1: WRITE REGISTER-7 ; 2 - 1: </pre>

```

;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE

DEFB 01111110B

; PROGRAM WRITE REGISTER-7 (SLDC FLAG)
; BIT 0 - 0: SYNC 8
;      1 - 1: SYNC 9
;      2 - 1: SYNC 10
;      3 - 1: SYNC 11
;      4 - 1: SYNC 12
;      5 - 1: SYNC 13
;      6 - 1: SYNC 14
;      7 - 0: SYNC 15

```

```

;*****
;
;EJECT
;*****
;PROCEDURE XMITCA TRANSMIT PROCEDURE FOR CHANNEL-A
;
;      THE PURPOSE OF THIS PROCEDURE IS TO TRANSMIT A
;      FRAME OF DATA OUT THE NETWORK CHANNEL-A PORT.
;
;INPUT - THIS PROCEDURE EXPECTS TWO INPUT PARAMETERS: THE
;        ADDRESS OF THE FIRST BYTE TO BE OUTPUT, AND THE NUMBER
;        OF BYTES TO BE SENT.
;
;PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE IX REG FOR
;             NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;             THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;             WILL BE OFFSET FROM THIS BASE LOCATION.
;             THE NEXT SECTION LOADS THE NUMBER OF BYTES TO BE
;             SENT AND THE ADDRESS OF THE FIRST BYTE FROM THE STACK.
;

```

```

; TRANSMIT ENABLE CODE IS THEN OUTPUT TO THE SIO,
; FOLLOWED BY A LOOP TO OUTPUT THE FRAME OF DATA.
; FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS
; RECOVERED, THE STACK DEALLOCATED, AND A RETURN EXECUTED.
;
; OUTPUT - A FRAME OF DATA IS SENT OUT NETWORK CHANNEL-A PORT.
;
; INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
; COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
; PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
; AND ARE RETREIVED WITH THE USE OF A BASE ADDRESS PLUS AN
; OFFSET.
;
; NOTES - 1. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
; PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;
; *****
; XMITCA: ; PROCEDURE TO TRANSMIT A FRAME OUT CH-A *****
;
; PUSH IX ; STORE IX REGISTER SET FOR RETURN
;
; LD IX,0 ; SET IX REGISTERS TO BASE OF STACK
; ADD IX,SP
;
; LD B,(IX+4) ; LD BYTES TO SEND
;
; LD L,(IX+6) ; LD ADDRESS OF DATA TO SEND
; LD H,(IX+7)
;
; LD C,A_DATA ; LD OUTGOING DATA PORT A ADDRESS
;
; LD A,10000000B ; RESET OUTGOING PORT TRANSMIT CRC GENERATOR
; ; BY WRITING TO WR0
; ; BIT 0 - 0: |
; ; 1 - 0: | WRITE REGISTER-0

```



```

;      2 - 0: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: |
;      7 - 1: | RESET Tx CRC GENERATOR

OUT (A_STAT),A

LD A,00000101B
; SET OUTGOING PORT COMMAND REGISTER TO 5
; BIT 0 - 1: |
;      1 - 0: | WRITE REGISTER-5
;      2 - 1: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE

OUT (A_STAT),A

LD A,11101011B
; PROGRAM WRITE REGISTER-5
; BIT 0 - 1: ENABLE Tx CRC
;      1 - 1: SET RTS
;      2 - 0: SDLC/CRC POLYNOMIAL
;      3 - 1: Tx ENABLE
;      4 - 0: SEND BREAK DISABLED
;      5 - 1: |
;      6 - 1: | Tx 8-BITS PER CHARACTER
;      7 - 1: | SET DTR

OUT (A_STAT),A

OUTI
DEC B
; OUTPUT FIRST BYTE
; DECREMENT BYTES TO SEND COUNTER

LD A,00000000B
; SET OUTGOING PORT COMMAND REGISTER TO 0
; BIT 0 - 0: |

```

```

;      1 - 0: | WRITE REGISTER-0
;      2 - 0: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 0: NULL CODE
;      7 - 0: NULL CODE
;
OUT (A_STAT),A

LD A,11000000B
; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
;      1 - 0: | WRITE REGISTER-0
;      2 - 0: |
;      3 - 0: NULL CODE
;      4 - 0: NULL CODE
;      5 - 0: NULL CODE
;      6 - 1: |
;      7 - 1: | RESET Tx UNDERRUN/EOM LATCH
;
OUT (A_STAT),A
;
TRNJ10 IN A,(A_STAT)
; READ OUTGOING PORT A STATUS REG
BIT 2,A
; AND WAIT UNTIL XMIT BUFFER IS EMPTY
JP Z,TRNJ10

TRNJ20 OUTI
; OUTPUT NEXT BYTE
DEC B
; DECREMENT BYTES TO SEND COUNTER
JP NZ,TRNJ10
; IF NUMBER OF BYTES > 0, SEND ANOTHER

POP IX
; RESTORE IX REGISTER SET
POP HL
; RECOVER RETURN ADDRESS

POP DE
; DEALLOCATE STACK
POP DE

```

JP (HL) ; RETURN TO CALLING PROGRAM

```

;*****
;
; *EJECT
;*****
;PROCEDURE XMITCB TRANSMITS A FRAME OF DATA OUT CHANNEL-B
;
; THE PURPOSE OF THIS PROCEDURE IS TO TRANSMIT A FRAME
; OF DATA OUT NETWORK CHANNEL-B PORT.
;
; INPUT - THIS PROCEDURE EXPECTS TWO INPUT PARAMETERS: THE
; ADDRESS OF THE FIRST BYTE TO BE OUTPUT, AND THE NUMBER OF BYTES
; TO BE SENT.
;
; PROCESSING - THIS PROCEDURE BEGINS WITH THE SAVE OF THE IX REGISTER
; FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES THE BASE
; LOCATION FOR THE STACK. ALL INPUT PARAMETERS WILL BE OFFSET
; FROM THIS BASE LOCATION.
; THE NEXT SECTION LOADS THE NUMBER OF BYTES TO BE SENT
; AND THE ADDRESS OF THE FIRST BYTE FROM THE STACK. TRANSMIT ENABLE
; CODE IS THEN OUTPUT TO THE SIO, FOLLOWED BY A LOOP TO OUTPUT THE
; FRAME OF DATA OUT NETWORK CHANNEL-B PORT.
; FINALLY, THE IX IS RESTORED, THE RETURN ADDRESS RECOVERED,
; THE STACK DEALLOCATED, AND A RETURN EXECUTED.
;
; OUTPUT - A FRAME OF DATA IS SENT OUT NETWORK CHANNEL-B PORT.
;
; INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
; COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT PARAMETERS
; ARE LOADED INTO THE STACK WITH A PUSH COMMAND AND ARE RETRIEVED
; WITH THE USE OF A BASE ADDRESS PLUS AN OFFSET.
;
; NOTES: 1. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01, PLZ USER GUIDE,
; SECTION 7 FOR DETAILS.
;

```

```
;*****;*****;*****;*****;*****;*****;*****;*****;*****;*****;*****;*****;  
XMITCB:  
  
    PUSH IX  
  
LD IX,0  
ADD IX,SP  
  
LD B,(IX+4)  
  
LD L,(IX+6)  
LD H,(IX+7)  
  
LD C,B_DATA  
  
LD A,1000000B  
  
OUT (B_STAT),A  
  
LD A,0000010B  
  
    ; SET PORT B COMMAND REGISTER TO 5  
    ; BIT 0 - 1: |  
    ; 1 - 0: | WRITE REGISTER-5  
    ; 2 - 1: |  
    ; 3 - 0: NULL CODE  
    ; 4 - 0: NULL CODE  
    ; 5 - 0: NULL CODE  
    ; 6 - 0: NULL CODE
```

```

OUT (B_STAT),A
LD A,11101011B
; 7 - 0: NULL CODE
; PROGRAM WRITE REGISTER-5
; BIT 0 - 1: Tx CRC ENABLE
; 1 - 1: SET RTS
; 2 - 0: SDLC/CRC
; 3 - 1: Tx ENABLE
; 4 - 0: SEND BREAK DISABLED
; 5 - 1: |
; 6 - 1: | Tx 8 BITS PER CHARACTER
; 7 - 1: SET DTR

OUT (B_STAT),A
OUTI
DEC B
LD A,00000000B
; OUTPUT FIRST BYTE
; DECREMENT OF BYTES TO SEND COUNTER
; SET OUTGOING COMMAND REGISTER TO 0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE
; 6 - 0: NULL CODE
; 7 - 0: NULL CODE

OUT (B_STAT),A
LD A,11000000B
; PROGRAM WRITE REGISTER-0
; BIT 0 - 0: |
; 1 - 0: | WRITE REGISTER-0
; 2 - 0: |
; 3 - 0: NULL CODE
; 4 - 0: NULL CODE
; 5 - 0: NULL CODE

```

```

;      6 - 1: |
;      7 - 1: | RESET Tx UNDERRUN/EOM LATCH
;
OUT (B_STAT),A
;
TRNJ30 IN A,(B_STAT)
; READ PORT B STATUS REGISTER
BIT 2,A
; AND WAIT UNTIL XMIT BUFFER IS EMPTY
JP Z,TRNJ30
;
TRNJ40 OUTI
; OUTPUT NEXT BYTE
DEC B
; DECREMENT BYTES TO SEND COUNTER
JP NZ,TRNJ30
; IF NUMBER BYTES > 0, SEND ANOTHER
POP IX
; RESTORE IX REGISTER SET
POP HL
; RECOVER RETURN ADDRESS
POP DE
; DEALLOCATE STACK
POP DE
;
JP (HL)
; RETURN TO CALLING PROGRAM
;
*****
;
; *EJECT
; *****
; PROCEDURE RECCHA SIO CHANNEL-A RECEIVE INTERRUPT CONTROLLER
;
; THE PURPOSE OF THIS PROCEDURE IS TO SERVICE NETWORK
; RECEIVE INTERRUPTS FROM CHANNEL-A.
;
; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. NT01NS IS
; THE NEXT SERVICE POSITION; NT01NE IS THE NEXT EMPTY POSITION;
; AND NT01SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
; NT01TB.
;

```

```

;PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
;PROGRAM'S REGISTERS. THE BYTE IS THEN INPUT FROM THE SIO CHANNEL-A.
;THE BYTE IS LOADED INTO THE NETWORK RECEIVE BUFFER (NC01TB) AND THE
;DATA POINTERS MODIFIED FOR WRAPAROUND IF NECESSARY. FINALLY
;THE INTERRUPTED PROGRAM'S REGISTERS ARE RESTORED,
;INTERRUPTS ENABLED, AND A RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE NT01TB AND
;THE NT01NE POSITION IS UPDATED TO REFLECT THE BYTE
;INSERTION.
;
;INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
;BY THE SIO CHANNEL-A. AS AN INTERRUPT IS IDENTIFIED, THE SIO
;DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;(IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;THE RECEIVE I/O INTERRUPT CONTROLLER.
;
;NOTES - NONE.
;*****
;***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
RECCHA: ; COMING FROM SIO CHANNEL-A.

EXTERNAL NT01TB NT01NE NT01SZ

EX AF,AF' ; SAVE REGS OF INTERRUPTED PROGRAM
EXX

IN A,(A_DATA) ; INPUT THE BYTE

LD DE,NT01TB ; SET HL TO NEXT EMPTY TABLE LOCATION
LD HL,(NT01NE)
ADD HL,DE

LD (HL),A ; LD BYTE INTO EMPTY TABLE LOCATION

```

```

LD HL,(NT01NE) ; LD EMPTY LOCATION POINTER AND
INC HL ; INCREMENT EMPTY LOCATION POINTER
LD (NT01NE),HL
LD DE,(NT01SZ) ; LD TABLE SIZE FOR CHECK
SBC HL,DE ; IF AT AND OF TABLE, RESET TO LOCATION ZERO
JR NZ,SRIJ10
LD HL,0
LD (NT01NE),HL

SRIJ10
EX AF,AF' ; RESTORE CALLING PROGRAM'S REGISTERS
EXX

EI ; ENABLE INTERRUPTS

RETI ; RETURN

```

```

;*****
;
;EJECT
;*****
;PROCEDURE RECCHB SIO CHANNEL-B INTERRUPT CONTROLLER
;
; THE PURPOSE OF THIS PROCEDURE IS TO SERVICE NETWORK
; RECEIVE INTERRUPTS FROM CHANNEL-B.
;
; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. NT02NS IS THE NEXT
; SERVICE POSITION; NT02NE IS THE NEXT EMPTY POSITION; AND NT02SZ IS
; THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE NT02TB.
;
;PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
; PROGRAM'S REGISTERS. THE BYTE IS THEN INPUT FROM THE SIO CHANNEL-B.
; THE BYTE IS LOADED INTO THE NETWORK RECEIVE TABLE (NC02TB) AND THE
; DATA POINTERS MODIFIED FOR WRAPAROUND IF NECESSARY. FINALLY THE

```



```

; INTERRUPTED PROGRAM'S REGISTERS ARE RESTORED, INTERRUPTS ENABLED,
; AND A RETURN FROM INTERRUPT EXECUTED.
;
; OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE NC02TB AND THE NT02NE
; POSITION IS UPDATED TO REFLECT THE BYTE INSERTION.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED BY
; THE SIO CHANNEL-B. AS AN INTERRUPT IS IDENTIFIED, THE SIO DEVELOPS
; AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE (IOVCTB). THIS OFFSET
; POSITION CONTAINS THE ADDRESS OF THE RECEIVE I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
; ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
; ; COMING FROM CHANNEL-B
; *****
RECCHB:
EXTERNAL NT02TB NT02NE NT02SZ
EX AF,AF' ; SAVE REGS OF INTERRUPTED PROGRAM
EXX
IN A,(B_DATA) ; INPUT THE BYTE
LD DE,NT02TB ; SET HL TO NEXT EMPTY TABLE LOCATION
LD HL,(NT02NE)
ADD HL,DE
LD (HL),A ; LD BYTE INTO EMPTY TABLE LOCATION
LD HL,(NT02NE) ; LD EMPTY LOCATION POINTER AND
INC HL ; INCREMENT EMPTY LOCATION POINTER
LD (NT02NE),HL
LD DE,(NT02SZ) ; LD TABLE SIZE FOR CHECK
SBC HL,DE ; IF AT END OF TABLE, RESET TO LOCATION ZERO

```

```

SRIJ20:
    JR NZ,SRIJ20
    LD HL,0
    LD (NT02NE),HL
    EX AF,AF'      ; RESTORE CALLING PROGRAM'S REGISTERS
    EXX
    EI              ; ENABLE INTERRUPTS
    RETI           ; RETURN

```

```

;*****
;
;EJECT
;*****
;PROCEDURE STCTC2      START CTC CHANNEL 2
;
;    THE PURPOSE OF THIS PROCEDURE IS TO START THE COUNTING OF
;    CTC CHANNEL-2. THIS PROCEDURE PROVIDES THE TIMING LOOP FOR
;    TRANSMITTING AN I-FRAME OF DATA ONTO THE NETWORK CHANNEL-A DATA LINK.
;
;    INPUT - NONE.
;
;PROCESSING - THIS PROCEDURE BEGINS WITH THE SAVING OF THE IX AND A REGISTERS
;              FOR NORMALIZATION OF RETURN. IT THEN SETS UP THE CTC CHANNEL-2
;              FOR INTERRUPT MODE AND LOADS IN A TIME CONSTANT. THIS TIME CONSTANT
;              IS 256 AT THE PRESENT TIME. THE PROCEDURE CONTINUES WITH THE
;              INITIALIZATION OF THE GLOBAL VARIABLE 'TIMCHA' TO FALSE.
;
;    AN EXAMPLE OF THE COMPLETE TIMING LOOP IS:
;              (Z-80 CLOCK) * TC * PRESCALAR * MAXNUM = TIME DELAY
;
;              ( 1 / (2.4x106) ) * 256 * 256 * 10 = 270 MILLISECONDS
;
;OUTPUT -      THE ONLY OUTPUT OF THIS PROCEDURE IS GLOBAL. IT IS THE

```

```

;
; VARIABLE 'TIMCHA' AND IS LOADED WITH THE VALUE TRUE UPON THE
; COMPLETION OF THE TIMING CYCLE.
;
; INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE TIME_DELAY IN THE
; MAIN MODULE.
;
; NOTES - 1. CAUTION SHOULD BE TAKEN IN INITIALIZING THE CONSTANTS FOR
; THE CTC OPERATION. BOTH THE TIME CONSTANT AND PRESCALAR SHOULD BE
; SET SO THAT THE TIME DURATION OF THE CYCLE ALLOWS SUFFICIENT TIME
; TO COMPLETE THE PROPER ROUTING OUT OF THE DATA FRAMES ONTO THE
; NETWORK.
;
; 2. SEE ZILOG DATA BOOK FOR THE Z8430 CTC OPERATION GUIDELINES.
;
; *****
STCTC2:      PUSH IX      ; SAVE THE IX REGISTER
             PUSH AF      ; SAVE THE AF REGISTER SET
             LD A,l0l0011b ; SET UP THE CTC CH-2 FOR INTERRUPT
             ; BIT 0 - 1: ENABLE INTERRUPT
             ; 1 - 1: TX INTERRUPT ENABLE
             ; 2 - 1: STATUS EFFECTS REGISTER
             ; 3 - 0: |
             ; 4 - 0: | RX INTERRUPT DISABLED
             ; 5 - 1: WAIT-READY
             ; 6 - 0: WAIT-READY
             ; 7 - 0: WAIT-READY
             OUT (CTC_2),A ; AND RESET. THEN OUTPUT TO CTC
             LD A,TC2      ; THEN LOAD AND OUTPUT THE
             OUT (CTC_2),A ; TIME CONSTANT FOR CHANNEL-A

             LD A,FALSE    ; PLACE VALUE FALSE IN ACCUMULATOR
             LD HL,TIMCHA  ; ADDRESS OF TIMCHA IN HL
             LD (TIMCHA),A ; MAKE TIMCHA = FALSE (INITIALIZATION)
             POP IX        ; RESTORE IX REGISTER
             POP AF        ; RESTORE AF REGISTER SET
             POP HL        ; OBTAIN THE RETURN ADDRESS
; *****

```

```

JP (HL)          ; RETURN TO CALLING PROCEDURE
*****
;
;EJECT
;*****
;PROCEDURE      STCTC3      START CTC CHANNEL 3
;
;      THE PURPOSE OF THIS PROCEDURE IS TO START THE COUNTING
;      OF CTC CHANNEL-3. THIS PROCEDURE PROVIDES THE TIMING LOOP FOR
;      TRANSMITTING AN I-FRAME OF DATA ONTO CHANNEL-B OF THE NETWORK
;      DATA LINK.
;
; INPUT - NONE
;
; PROCESSING - THIS PROCEDURE BEGINS WITH THE SAVING OF THE IX AND A REGISTERS
;              FOR NORMALIZATION OF RETURN. IT THEN SETS UP THE CTC CH NUMBER 3
;              FOR INTERRUPT MODE AND LOADS IN A TIME CONSTANT. THIS TIME
;              CONSTANT IS 256 AT THE PRESENT TIME. THE PROCEDURE CONTINUES WITH
;              THE INITIALIZATION OF THE GLOBAL VARIABLE 'TIMCHB' TO FALSE.
;
; AN EXAMPLE OF THE COMPLETE TIMING LOOP IS:
;      (Z-80 CLOCK)* TC * PRESCALAR * MAXNUM = TIME DELAY
;
;      ( 1 / (2.4X106) ) * 256 * 256 * 10 = 270 MILLISECONDS
;
; OUTPUT - THE ONLY OUTPUT OF THIS PROCEDURE IS GLOBAL. IT IS THE
;           VARIABLE 'TIMCHB' AND IS LOADED WITH THE VALUE TRUE UPON
;           THE COMPLETION OF THE TIMING CYCLE.
;
; INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE TIME_DELAY
;             IN THE MAIN MODULE.
;
; NOTES - 1. CAUTION SHOULD BE TAKEN IN INITIALIZING THE CONSTANTS FOR
;           THE CTC OPERATION. BOTH THE TIME CONSTANT AND PRESCALAR

```

SHOULD BE SET SO THAT THE TIME DURATION OF THE CYCLE ALLOWS
SUFFICIENT TIME TO COMPLETE THE PROPER ROUTING OUT OF THE
DATA FRAMES.

2. SEE ZILOG DATA BOOK FOR THE Z8430 CTC OPERATION GUIDELINES.

```

;
;
;
;
;
; *****
STCTC3: *****
        PUSH IX          ; SAVE THE IX REGISTER *****
        PUSH AF          ; SAVE THE AF REGISTER SET *****
        LD A, 10100111B  ; SET UP THE CTC 3 FOR INTERRUPT *****
        ; BIT 0 - 0: ENABLE INTERRUPT *****
        ; 1 - 1: TX INTERRUPT ENABLE *****
        ; 2 - 1: STATUS EFFECTS REGISTER *****
        ; 3 - 0: | *****
        ; 4 - 0: | RX INTERRUPT DISABLE *****
        ; 5 - 1: WAIT-READY *****
        ; 6 - 0: WAIT-READY *****
        ; 7 - 1: WAIT-READY *****
        OUT (CTC_3), A    ; AND RESET. THEN OUTPUT TO CTC *****
        LD A, TC3        ; THEN LOAD AND OUTPUT THE *****
        OUT (CTC_3), A    ; TIME CONSTANT *****

        LD A, FALSE      ; PLACE VALUE FALSE IN ACCUMULATOR *****
        LD HL, TIMCHB    ; ADDRESS OF TIMCHB IN HL *****
        LD (TIMCHB), A    ; MAKE TIMCHB = FALSE (INITIALIZATION) *****
        POP IX           ; RESTORE IX REGISTER *****
        POP AF           ; RESTORE AF REGISTER SET *****
        POP HL           ; OBTAIN THE RETURN ADDRESS *****
        JP (HL)          ; RETURN TO CALLING PROCEDURE *****
; *****
; *****

```

[illegible]

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE UNINID NETWORK OPERATING SYSTEM (N.OS) WITH THE MAIN LINE OF PROCESSING. THE N.OS IS REQUIRED TO INPUT/OUTPUT DATA PASSED TO IT FROM FOUR LOCAL CHANNELS OR RECEIVED FROM EITHER NETWORK CHANNEL A OR B.

THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE 'MAIN', AND PROCEDURES INIT_N_TAB, DET_DEST_ONE, DET_DEST_TWO, DET_DEST_LN, LLD_TAB_HSKP, SRVC_TAB_HSKP, BUILD_I_FRAME, BUILD_S_FRAME, ROUTE_IN, TIME_DELAY_CHA, TIME_DELAY_CHB, AND ROUTE_OUT.

```
*****MAIN MODULE*****!
```

TYPE

РВУТЕ ^ВУТЕ

CONSTANT ! CONSTANTS FOR THE NETWORK SIDE OF THE UNID !

```

FALSE := 0      ! USE AS FLAGS TO TEST !
TRUE := NOT FALSE ! BITS FOR BRANCHING !
CONCTC := %D5 ! NETWORK MONITOR CTC PORT ADDRESS !
CONCMD := %DF ! NETWORK MONITOR USART COMMAND PORT ADDRESS !
CONDAT := %DE ! NETWORK MONITOR USART DATA PORT ADDRESS !
NET_RI_DEST_ERR := 10 ! NET ROUTE_IN DEST ERROR ENTRY !
NET_RO_DEST_ERR := 11 ! NET ROUTE_OUT DEST ERROR ENTRY !
PPACKET_SIZE := 133 ! PACKET IS 133-BYTE BLOCK !
FRAME_SIZE := PACKET_SIZE + 2 ! 2-BYTE HEADER FOR A FRAME !
PACKETS_IN_TABLE := 10
PACKET_TABLE_SIZE := PACKET_SIZE * PACKETS_IN_TABLE
FRAMES_IN_TABLE := PACKETS_IN_TABLE ! ONE PACKET PER FRAME !
FRAME_TABLE_SIZE := FRAME_SIZE * FRAMES_IN_TABLE

```

```

! NETWORK VARIABLES FOR THIS UNID !
THIS_UNID_NBR := %02 ! UNIQUE ADDRESS OF THIS UNID !
THIS_COUNTRY_CODE := %01 ! COUNTRY WHERE THIS UNID RESIDES !
MAX_COUNTRY_CODE := %01 ! COUNTRIES CURRENTLY OPERATIONAL !
MAX_NETWORK_CODE := 3 ! NUMBER OF UNIDS OPERATIONAL IN CC !
STAT_NBR := 20 ! MBER OF ENTRIES IN STATUS TABLE !

GLOBAL
! VARIABLES USED IN N.MAIN_U2 AND N.INSIO_U2 !
CTCNOA BYTE := 00 ! PROGRESSIVE NUMBER OF LOOP COUNTS
FOR NETWORK CHANNEL-A !
CTCNOB BYTE := 00 ! PROGRESSIVE NUMBER OF LOOP COUNTS
FOR NETWORK CHANNEL-B !
MAXNOA BYTE := %64 ! MAXIMUM NUMBER OF COUNTING LOOPS
FOR NETWORK CHANNEL-A !
MAXNOB BYTE := %64 ! MAXIMUM NUMBER OF COUNTING LOOPS
FOR NETWORK CHANNEL-B !
TIMCHA BYTE := TRUE ! FLAG FOR CH-A WAIT LOOP !
TIMCHB BYTE := TRUE ! FLAG FOR CH-B WAIT LOOP !

EXTERNAL ! IN N.INSIO_U2 !
INSIO PROCEDURE
XMITCA PROCEDURE (SRCADD PBYTE, NUMBYT WORD)
XMITCR PROCEDURE (SRCADD PBYTE, NUMBYT WORD)
STCTC2 PROCEDURE
STCTC3 PROCEDURE

EXTERNAL ! IN U.LIB_U2 !
MOVSEQ PROCEDURE (SRCADD PBYTE, DTDADD PBYTE, NUMBYT BYTE)
SNDSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)
RECSEQ PROCEDURE (CMDPRT BYTE, DATPRT BYTE, BYTADD PBYTE, NUMBYT BYTE)

EXTERNAL ! IN N.TAB_U2 !
INIT_N_TAB PROCEDURE

```

NT01TB ARRAY [FRAME_TABLE_SIZE BYTE]
NT01NS INTEGER
NT01NE INTEGER
NT01SZ INTEGER

NT02TB ARRAY [FRAME_TABLE_SIZE BYTE]
NT02NS INTEGER
NT02NE INTEGER
NT02SZ INTEGER

OUTFRAME_CHA_TB ARRAY [FRAME_TABLE_SIZE BYTE]
OUTFRAME_CHA_NS INTEGER
OUTFRAME_CHA_NE INTEGER
OUTFRAME_CHA_SZ INTEGER

OUTFRAME_CHB_TB ARRAY [FRAME_TABLE_SIZE BYTE]
OUTFRAME_CHB_NS INTEGER
OUTFRAME_CHB_NE INTEGER
OUTFRAME_CHB_SZ INTEGER

EXTERNAL ! IN U.SHTAB_U2 !

LCNTTB ARRAY [PACKET_TABLE_SIZE BYTE]
LCNTNS INTEGER
LCNTNE INTEGER
LCNTSZ INTEGER

NTLCTB ARRAY [PACKET_TABLE_SIZE BYTE]
NTLCNS INTEGER
NTLCNE INTEGER
NTLCSZ INTEGER

STATTB ARRAY [STAT_NBR BYTE]

INTERNAL ! INTERNAL VARIABLES USED ON THE NETWORK SIDE !

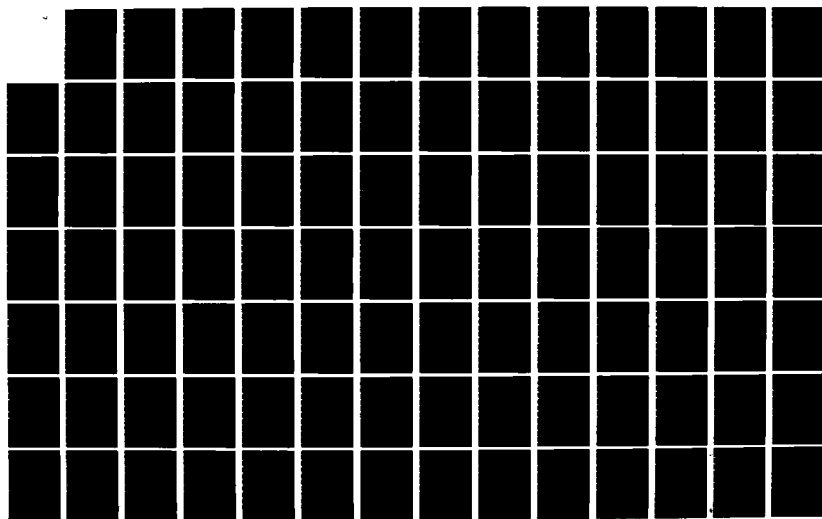
AD-A138 119

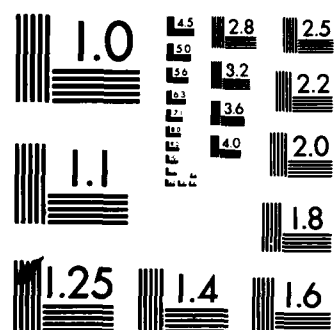
PROTOCOL STANDARDS AND IMPLEMENTATION WITHIN THE
DIGITAL ENGINEERING LABO... (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... P W PHISTER
DEC 83 AFIT/GE/EE/83D-58 F/G 17/2

3/4

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

DESTINATION WORD ! DESTINATION OF A FRAME !

! VARIABLES FOR NETWORK CHANNEL-A !
SEQ_BIT_A BYTE := FALSE ! MOD-2 SEQUENCE BIT !
ACKNOWLEDGE_A BYTE := FALSE ! ID FOR GOOD ACK FRAME !

! VARIABLES FOR NETWORK CHANNEL-B !
SEQ_BIT_B BYTE := FALSE ! MOD-2 SEQUENCE BIT !
ACKNOWLEDGE_B BYTE := FALSE ! ID FOR GOOD ACK FRAME !

! VARIABLES FOR BOTH NETWORK CHANNELS !
INPUT_SEQ_BIT BYTE := FALSE ! SEQ BIT GOING INTO FRAME !
THIS_SEQ_BIT BYTE := FALSE ! CURRENT SEQUENCE BIT !

INTERNAL ! TEST POINTS USED AS AN AID TO DEBUGGING !

STARTUP_HDR ARRAY [*BYTE] := '%R%L'
'UNID 2 NETWORK OS%R%L'
'VERSION 27 SEP 83%R%L'
'EXECUTING%R%L'

```

```

TP_1 ARRAY[*BYTE] := '%R%LTP_1: ENTERING INIT_N_TAB PROCEDURE'
TP_2 ARRAY[*BYTE] := '%R%LTP_2: ENTERING INSIO PROCEDURE'
TP_3 ARRAY[*BYTE] := '%R%LTP_3: STARTING ROUTE_IN-ROUTE_OUT LOOP'
TP_4 ARRAY[*BYTE] := '%R%LTP_4: ENTERING ROUTE_IN PROCEDURE'
TP_5 ARRAY[*BYTE] := '%R%LTP_5: INCOMING DATA LOCATED IN NETWORK CHANNEL-A'
TP_6 ARRAY[*BYTE] := '%R%LTP_6: DATA IS NTOITB-TO-OUTFRAME_CHB_TB TRANSFER'
TP_7 ARRAY[*BYTE] := '%R%LTP_7: DATA IS DESTINED FOR THIS UNID'
TP_8 ARRAY[*BYTE] := '%R%LTP_8: DATA IS INCOMING S-FRAME'
TP_9 ARRAY[*BYTE] := '%R%LTP_9: DATAS SEQ_BIT_A = 1'
TP_10 ARRAY[*BYTE] := '%R%LTP_10: DATAS SEQ_BIT_A = 0'
TP_11 ARRAY[*BYTE] := '%R%LTP_11: DATA IS INCOMING I-FRAME'
TP_12 ARRAY[*BYTE] := '%R%LTP_12: DATAS SEQ_BIT_A = 1'
TP_13 ARRAY[*BYTE] := '%R%LTP_13: DATAS SEQ_BIT_A = 0'

```

TP_14	ARRAY[*BYTE]	::'%R%LTP_14:	INCOMING DATA LOCATED IN NETWORK CHANNEL-B'
TP_15	ARRAY[*BYTE]	::'%R%LTP_15:	DATA IS NT02TB-TO-OUTFRAME_CHA_TB TRANSFER'
TP_16	ARRAY[*BYTE]	::'%R%LTP_16:	DATA IS DESTINED FOR THIS UNID'
TP_17	ARRAY[*BYTE]	::'%R%LTP_17:	DATA IS INCOMING S-FRAME'
TP_18	ARRAY[*BYTE]	::'%R%LTP_18:	DATAS SEQ_BIT_B = 1'
TP_19	ARRAY[*BYTE]	::'%R%LTP_19:	DATAS SEQ_BIT_B = 0'
TP_20	ARRAY[*BYTE]	::'%R%LTP_20:	DATA IS INCOMING I-FRAME'
TP_21	ARRAY[*BYTE]	::'%R%LTP_21:	DATAS SEQ_BIT_B = 1'
TP_22	ARRAY[*BYTE]	::'%R%LTP_22:	DATAS SEQ_BIT_B = 0'
TP_23	ARRAY[*BYTE]	::'%R%LTP_23:	DATA LOCATED IN LCNTTB'
TP_24	ARRAY[*BYTE]	::'%R%LTP_24:	INCOMING PACKET DESTINED FOR CHANNEL-A'
TP_25	ARRAY[*BYTE]	::'%R%LTP_25:	INCOMING PACKET DESTINED FOR CHANNEL-B'
TP_26	ARRAY[*BYTE]	::'%R%LTP_26:	OUTGOING DATA LOCATED IN NETWORK CHANNEL-A'
TP_27	ARRAY[*BYTE]	::'%R%LTP_27:	DATA IS OUTFRAME_CHA_TB-TO-CHANNEL-A TRANSFER'
TP_28	ARRAY[*BYTE]	::'%R%LTP_28:	OUTGOING DATA LOCATED IN NETWORK CHANNEL-B'
TP_29	ARRAY[*BYTE]	::'%R%LTP_29:	DATA IS OUTFRAME_CHB_TB-TO-CHANNEL-B TRANSFER'
TP_30	ARRAY[*BYTE]	::'%R%LTP_30:	DESTINATION OF NT01TB IS NL'
TP_31	ARRAY[*BYTE]	::'%R%LTP_31:	DESTINATION OF NT01TB IS NN'
TP_32	ARRAY[*BYTE]	::'%R%LTP_32:	DESTINATION OF NT02TB IS NL'
TP_33	ARRAY[*BYTE]	::'%R%LTP_33:	DESTINATION OF NT02TB IS NN'
TP_34	ARRAY[*BYTE]	::'%R%LTP_34:	DESTINATION_UNID >= THIS_UNID_NBR'
TP_35	ARRAY[*BYTE]	::'%R%LTP_35:	DESTINATION_UNID < THIS_UNID_NBR'
TP_36	ARRAY[*BYTE]	::'%R%LTP_36:	DESTINATION IS CHANNEL-A'
TP_37	ARRAY[*BYTE]	::'%R%LTP_37:	DESTINATION IS CHANNEL-B'
TP_38	ARRAY[*BYTE]	::'%R%LTP_38:	BUILDING I-FRAME FOR CHANNEL-A'
TP_39	ARRAY[*BYTE]	::'%R%LTP_39:	BUILDING I-FRAME FOR CHANNEL-B'
TP_40	ARRAY[*BYTE]	::'%R%LTP_40:	BUILDING S-FRAME FOR CHANNEL-A'
TP_41	ARRAY[*BYTE]	::'%R%LTP_41:	BUILDING S-FRAME FOR CHANNEL-B'
TP_42	ARRAY[*BYTE]	::'%R%LTP_42:	ENTERING ROUTE_OUT PROCEDURE'
TP_43	ARRAY[*BYTE]	::'%R%LTP_43:	END OF ROUTE_IN-ROUTE_OUT LOOP'
TP_44	ARRAY[*BYTE]	::'%R%LTP_44:	HAVE EXITED ROUTE_IN-ROUTE_OUT LOOP'
TP_45	ARRAY[*BYTE]	::'%R%LTP_45:	DATA IS NT01TB-TO-NTLCTB TRANSFER'
TP_46	ARRAY[*BYTE]	::'%R%LTP_46:	DATA IS NT02TB-TO-NTLCTB TRANSFER'
TP_47	ARRAY[*BYTE]	::'%R%LTP_47:	ERROR-UNID DESIGNATION > MAX_NETWORK_CODE'
TP_48	ARRAY[*BYTE]	::'%R%LTP_48:	ERROR-UNID DESIGNATION > MAX_NETWORK_CODE'

```

TP_49 ARRAY[*BYTE] := '%R&LTP_49: DATAS SEQ_BIT_A = 1'
TP_50 ARRAY[*BYTE] := '%R&LTP_50: DATAS SEQ_BIT_A = 0'
TP_51 ARRAY[*BYTE] := '%R&LTP_51: DATAS SEQ_BIT_B = 1'
TP_52 ARRAY[*BYTE] := '%R&LTP_52: DATAS SEQ_BIT_B = 0'

```

INTERNAL

```

!*****
PROCEDURE DET_DEST_ONE DETERMINE DEST OF INCOMING FRAME FROM CHANNEL-A

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE
DESTINATION OF A SPECIFIED INCOMING FRAME COMING INTO CHANNEL-A.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
THE TABLE LOCATION OF THE FRAME TO BE EVALUATED.

PROCESSING - THIS PROCEDURE LOOKS AT THE FIRST BYTE OF THE
NETWORK INPUT TABLE (NT01TB) SERVICING CH-A AND MAKES THE DETERMINATION
IF THE DATA FRAME IS DESTINED FOR THIS PARTICULAR UNID OR
SOME OTHER UNID. IF IT GOES TO THIS UNID THEN 'NL', IF TO
SOME OTHER UNID THEN 'NN'.

OUTPUT - THE PROCEDURE OUTPUTS A TWO CHARACTER ASCII VALUE
INDICATING THE TABLE OR CHANNEL DESTINATION OF THE FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN FOR INPUT
FRAMES.

```

NOTES - 1. FOR THIS PROCEDURE, S-FRAMES DESTINED FOR THIS UNID ARE 'NL'
*****

```

INTERNAL

```

DET_DEST_ONE PROCEDURE (TABLE WORD)
RETURNS (DESTINATION WORD)

```

```

IF ((NT01TB [NT01NS] AND %F0) / %10) = THIS_UNID_NBR
THEN
    SNDSEQ (CONCMD, CONDAT, TP_30[0], 36)
    ! SEND TEST_POINT_30 MESSAGE TO NETWORK MONITOR !
    DESTINATION := 'NL'
ELSE
    SNDSEQ (CONCMD, CONDAT, TP_31[0], 36)
    ! SEND TEST_POINT_31 MESSAGE TO NETWORK MONITOR !
    DESTINATION := 'NN'
FI

```

******* ! *******

! *****

PROCEDURE DET DEST TWO DETERMINE DEST OF INCOMING FRAME FROM CH-B OF NET *****

INPUT - THE INPUT IS A TWO ASCII CHARACTER ASCII VALUE INDICATING THE TABLE LOCATION OF THE FRAME TO BE EVALUATED.

PROCESSING - THIS PROCEDURE LOOKS AT THE FIRST BYTE OF THE NETWORK INPUT TABLE (NT02TB SERVICING CH-B AND MAKES THE DETERMINATION IF THE DATA FRAME IS DESTINED FOR THIS PARTICULAR UNID OR ANOTHER UNID ON THE NETWORK. IF IT GOES TO THIS UNID THEN 'NL', IF TO ANOTHER UNID THEN 'NN'.

- THIS PROCEDURE OUTPUTS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE OR OUTPUT CHANNEL DESTINATION OF THE FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE IN FOR INPUT FRAMES.

NOTES - 1. FOR THIS PROCEDURE S-FRAMES DESTINED FOR THIS UNID ARE 'NL'.

 INTERNAL
 *****!

DET_DEST_TWO PROCEDURE (TABLE WORD)
 RETURNS (DESTINATION WORD)

ENTRY

```

IF ((NT02TB [NT02NS] AND %F0) / %10) = THIS_UNID_NBR
  THEN
    SNDSEQ(CONCMD,CONDAT, TP_32[0],36)
    ! SEND TEST_POINT_32 MESSAGE TO NETWORK MONITOR !
    DESTINATION := 'NL'
  ELSE
    SNDSEQ(CONCMD,CONDAT, TP_33[0],36)
    ! SEND TEST_POINT_33 MESSAGE TO NETWORK MONITOR !
    DESTINATION := 'NN'
  FI

```

END DET_DEST_TWO

*****!
 *****!
 PROCEDURE DET_DEST_LN DETERMINES DESTINATION OF AN OUTGOING PACKET
 ONTO THE NETWORK

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE OUTGOING
 NETWORK CHANNEL FOR THE PACKET LOCATED IN TABLE LCNTTB.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE
 TABLE REQUIRING EVALUATION.


```

        SNDSEQ(CONCMD,CONDAT, TP_35[0],41)
        ! SEND TEST_POINT_35 MESSAGE TO NETWORK MONITOR !
        DISTANCE_LEFT := INTEGER(THIS_UNID_NBR) +
            (MAX_NETWORK_CODE-DESTINATION_UNID+1)
        DISTANCE_RIGHT :=DESTINATION_UNID-INTEGER(THIS_UNID_NBR)
    FI
    IF DISTANCE_LEFT >= DISTANCE_RIGHT
    THEN
        SNDSEQ(CONCMD,CONDAT, TP_36[0],33)
        ! SEND TEST_POINT_36 MESSAGE TO NETWORK MONITOR !
        DESTINATION := 'CA'
    ELSE
        SNDSEQ(CONCMD,CONDAT, TP_37[0],33)
        ! SEND TEST_POINT_37 MESSAGE TO NETWORK MONITOR !
        DESTINATION := 'CB'
    FI

```

END DET_DEST_LN

!*****!

!*****!

PROCEDURE LD_TAB_HSKP LOAD TABLE HOUSEKEEP *****

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
TABLE AFTER THE LOADING OF A PACKET OR A FRAME.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT-EMPTY-BYTE POINTER BY A PACKET OR FRAME
SIZE, AND ADJUSTS FOR TABLE WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-EMPTY-BYTE

POINTER ADVANCED BY THE LENGTH OF A PACKET OR A FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

NOTES - NONE.

INTERNAL

LD_TAB_HSKP PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE '01' ! IF CALLED TO HSKP INCOMING NET CH-A TABLE !

THEN

NT01NE := NT01NE + FRAME_SIZE

IF NT01NE >= NT01SZ ! IF TABLE WRAP !

THEN

NT01NE := 0

FI

CASE '02' ! IF CALLED TO HSKP INCOMING NET CH-B TABLE !

THEN

NT02NE := NT02NE + FRAME_SIZE

IF NT02NE >= NT02SZ ! IF TABLE WRAP !

THEN

NT02NE := 0

FI

CASE '03' ! IF CALLED TO HSKP OUTGOING NET CH-A TABLE !

THEN

OUTFRAME_CHA_NE := OUTFRAME_CHA_NE + FRAME_SIZE

IF OUTFRAME_CHA_NE >= OUTFRAME_CHA_SZ

THEN

```

        OUTFRAME_CHA_NE := 0
    FI

    CASE '04' ! IF CALLED TO HSKP OUTGOING NET CH-B TABLE !
    THEN
        OUTFRAME_CHB_NE := OUTFRAME_CHB_NE + FRAME_SIZE
        IF OUTFRAME_CHB_NE >= OUTFRAME_CHB_SZ
        THEN
            OUTFRAME_CHB_NE := 0
        FI
    FI

```

```

    CASE 'NL' ! IF CALLED TO HSKP NETWORK-TO-LOCAL TABLE !
    THEN
        NTLCNE := NTLCNE + PACKET_SIZE
        IF NTLCNE >= NTLCNZ ! IF TABLE WRAP !
        THEN
            NTLCNE := 0
        FI
    FI

```

```

    CASE 'LN' ! IF CALLED TO HSKP LOCAL-TO-NETWORK TABLE !
    THEN
        LCNTNE := LCNTNE + PACKET_SIZE
        IF LCNTNE >= LCNTSZ ! IF TABLE WRAP !
        THEN
            LCNTNE := 0
        FI
    FI

```

```

    FI ! END OF IF TABLE CONDITION !

```

```

END LD_TAB_HSKP

```

```

!*****!
!*****!

```

PROCEDURE SRVC_TAB_HSKP SERVICE TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
TABLE AFTER SERVICING (REMOVING EITHER A PACKET OR FRAME).

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING
THE TABLE REQUIRING HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT-BYTE-TO-BE-SERVED POINTER BY A PACKET OR
FRAME SIZE, AND ADJUSTS FOR TABLE WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-BYTE-TO-BE-SERVED
POINTER ADVANCED BY THE LENGTH OF A PACKET OR FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN AND ROUTE_OUT.

NOTES - NONE.

INTERNAL

SRVC_TAB_HSKP PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE '01' ! IF CALLED TO HSKP INCOMING NET CH-A TABLE !
THEN

NT01NS := NT01NS + FRAME_SIZE

IF NT01NS >= NT01SZ ! IF TABLE WRAP !

THEN

NT01NS := 0

FI

```

CASE '02' ! IF CALLED TO HSKP INCOMING NET CH-B TABLE !
THEN
  NT02NS := NT02NS + FRAME_SIZE
  IF NT02NS >= NT02SZ ! IF TABLE WRAP !
  THEN
    NT02NS := 0
  FI

```

```

CASE '03' ! IF CALLED TO HSKP OUTGOING NET CH-A TABLE !
THEN
  OUTFRAME_CHA_NS := OUTFRAME_CHA_NS + FRAME_SIZE
  IF OUTFRAME_CHA_NS >= OUTFRAME_CHA_SZ
  THEN
    OUTFRAME_CHA_NS := 0
  FI

```

```

CASE '04' ! IF CALLED TO HSKP OUTGOING NET CH-B TABLE !
THEN
  OUTFRAME_CHB_NS := OUTFRAME_CHB_NS + FRAME_SIZE
  IF OUTFRAME_CHB_NS >= OUTFRAME_CHB_SZ
  THEN
    OUTFRAME_CHB_NS := 0
  FI

```

```

CASE 'LN' ! IF CALLED TO HOUSEKEEP LOCAL-TO-NET TABLE !
THEN
  LCNTNS := LCNTNS + PACKET_SIZE
  IF LCNTNS >= LCNTSZ ! IF TABLE RAFL !
  THEN
    LCNTNS := 0
  FI

```

```

FI ! END OF IF TABLE CONDITION !

```

```

END SRVC_TAB_HSKP

```

*****!
*****!
PROCEDURE BUILD_I_FRAME PROCEDURE FOR TRANSFORMING A PACKET INTO AN I-FRAME

THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM A PACKET OF DATA DELIVERED FROM THE LOCAL SIDE OF THE UNID INTO AN I-FRAME TO BE PLACED INTO ON OF THE OUTGOING NET TABLES. THIS PROCEDURE ADDS ON THE TWO HEADER BYTES: ADDRESS BYTE AND CONTROL BYTE.

INPUT - THIS PROCEDURE RECEIVES A TWO CHARACTER ASCII VALUE INDICATING THE LOCATION OF THE OUTGOING NETWORK PORT WHERE THE NEWLY CREATED I-FRAME IS TO BE PLACED.

PROCESSING - THE PROCESSING BEGINS WITH THE PASSING OF THE TABLE WHERE THE NEWLY CREATED I-FRAME IS TO BE PLACED. THE DESTINATION ADDRESS IS READ FROM THE PACKET HEADER LOCATED IN TABLE LCNTTB AND PLACED INTO THE FRAME HEADER FIELD. THE SOURCE ADDRESS IS THEN ADDED BY USING THE THIS_UNID_NBR VARIABLE. THE CONTROL FIELD BYTE IS INITIALIZED TO ZERO AND WILL BE CHANGED BASED UPON LATER ACTION BY THE ROUTING PROCEDURE.

OUTPUT - THIS PROCEDURE PLACES THE FIRST TWO BYTES INTO THE INDICATED OUTPUT TABLE BEFORE THE PACKET IS TRANSFERRED OVER TO THE CORRECT OUTPUT NETWORK TABLE.

INTERFACE - THIS PROCEDURE IS CALLED FROM THE ROUTE_IN PROCEDURE FOR THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.

NOTES - 1. THE HEADER INFORMATION IS BASED ON THE FIELDS DEFINED BY THIS THESIS ONLY. EVEN THOUGH THEY CORRESPOND TO THE HDLC PROTOCOL, CAUTION SHOULD BE TAKEN BEFORE ADDING OR DELETING ADDITIONAL HEADERS.

*****!

INTERNAL

BUILD_I_FRAME PROCEDURE (TABLE WORD)

LOCAL

ADDRESS_BYTE BYTE
CONTROL_BYTE BYTE

ENTRY

ADDRESS_BYTE := (LCNTTB[LCNTNS+0] AND %F0) OR (THIS_UNID_NBR)
CONTROL_BYTE := 00
IF TABLE

CASE 'CA' ! I-FRAME IS GOING OUT NETWORK CHANNEL-A !
THEN
 SNDSEQ(CONCMD, CONDAT, TP_38[0], 39)
 ! SEND TEST_POINT_38 MESSAGE TO NETWORK MONITOR !
 OUTFRAME_CHA_TB [OUTFRAME_CHA_NE+0] := ADDRESS_BYTE
 OUTFRAME_CHA_TB [OUTFRAME_CHA_NE+1] := CONTROL_BYTE
 MOVSEQ (LCNTTB[LCNTNS],
 OUTFRAME_CHA_TB [OUTFRAME_CHA_NE+2], PACKET_SIZE)
 LD_TAB_HSKP ('03')

CASE 'CB' ! I-FRAME IS GOING OUT NETWORK CHANNEL-B !
THEN
 SNDSEQ(CONCMD, CONDAT, TP_39[0], 39)
 ! SEND TEST_POINT_39 MESSAGE TO NETWORK MONITOR !
 OUTFRAME_CHB_TB [OUTFRAME_CHB_NE+0] := ADDRESS_BYTE
 OUTFRAME_CHB_TB [OUTFRAME_CHB_NE+1] := CONTROL_BYTE
 MOVSEQ (LCNTTB[LCNTNS],
 OUTFRAME_CHB_TB [OUTFRAME_CHB_NE+2], PACKET_SIZE)
 LD_TAB_HSKP ('04')

FI ! END OF IF TABLE CONDITION !

END BUILD_I_FRAME

!*****! !*****! !*****!

!*****! !*****! !*****!
PROCEDURE BUILD_S_FRAME BUILD A SUPERVISORY FRAME

THE PURPOSE OF THIS PROCEDURE IS TO BUILD A SUPERVISORY FRAME
AND PLACE IT INTO THE APPROPRIATE OUTGOING TABLE FOR TRANSMISSION TO
THE NETWORK VIA EITHER NETWORK CHANNEL-A OR CHANNEL-B.

INPUT - THIS PROCEDURE IS PASSED A TWO CHARACTER ASCII VALUE INDICATING
THE LOCATION OF THE OUTGOING S-FRAME AND THE SEQUENCE BIT OF
THE INCOMING I-FRAME (MODULO 2).

PROCESSING - THE PROCESSING BEGINS WITH THE PASSING OF THE TABLE WHERE
THE NEWLY CREATED S-FRAME IS TO BE PLACED. THE PROCEDURE THEN
PERFORMS A SERIES OF LOGICAL OPERATIONS AND DIVISIONS TO SWAP THE
HIGH 4 ORDER BITS OF THE ADDRESS WORD WITH THE LOW ORDER BITS.
THIS SIMPLY INTERCHANGES THE DESTINATION AND SOURCE ADDRESSES.
IT THEN SETS THE CONTROL WORD ACCORDING TO THE SEQUENCE BIT
OF THE INCOMING I-FRAME. THESE BYTES ARE THEN PLACED IN
THE FIRST AND SECOND HEADER POSITIONS OF THE OUTGOING NETWORK TABLE.
APPROPRIATE TABLE POINTERS ARE THEN UPDATED.

OUTPUT - SEE PROCESSING

INTERFACE - THIS PROCEDURE IS CALLED BY ROUTE-IN WHENEVER A NEW I-FRAME
ARRIVES.

NOTES - 1. THE X.25 AND LAP PROTOCOLS ALLOW FOR ADDITIONAL TYPES OF
S-FRAMES WHICH MAY BE INCORPORATED DURING FURTHER DEVELOPMENT.


```
*****INTERNAL*****!
```

```
BUILD_S_FRAME PROCEDURE (TABLE_WORD, INPUT_SEQ_BIT_BYTE)
```

```
LOCAL
```

```
ADDRESS_WORD BYTE
CONTROL_WORD BYTE
INDEX INTEGER
```

```
ENTRY
```

```
IF TABLE
```

```
CASE '01' ! IF CALLED TO BUILD S_FRAME FOR CH-A !
THEN
  SNDSEQ(CONCMD, CONDAT, TP_40[0], 39)
  ! SEND TEST_POINT_40 MESSAGE TO NETWORK MONITOR !
  ADDRESS_WORD := (((NT01TB [NT01NS] AND %0F) *
                    %10) OR (((NT01TB [NT01NS] AND
                    %F0) / %10))
  ! MASK OFF RIGHT 4-BITS AND SHIFT RIGHT. MASK OFF
  ! LEFT 4-BITS AND SHIFT LEFT. THEN COMBINE !
  IF INPUT_SEQ_BIT = TRUE
  THEN CONTROL_WORD := %A0
  ! IF SEQ_BIT_A = 1 THEN CONTROL_WORD = 10100000 !
  ELSE CONTROL_WORD := %80
  ! IF SEQ_BIT_A = 0 THEN CONTROL_WORD = 10000000 !
FI
INDEX := 0
DO
  OUTFRAME_CHA_TB [OUTFRAME_CHA_NE + INDEX] := 0
  INDEX += 1
  IF INDEX > 134 THEN EXIT FI
```

```

OD ! CLEAR ALL DATA FOR TRANSMISSION OF S-FRAME !
OUTFRAME_CHA_TB[OUTFRAME_CHA_NE+0] := ADDRESS_WORD
OUTFRAME_CHB_TB[OUTFRAME_CHB_NE+1] := CONTROL_WORD
! MOVE DATA TO OUTGOING TABLE FOR XMISSION OUT CH-A !
LD_TAB_HSKP('03')
! HOUSEKEEP THE OUTFRAME_CHA_TB TABLE !

CASE '02' ! IF CALLED TO BUILD S_FRAME FOR CH-B !
THEN
  SNDSEQ(CONCMD, CONDAT, TP_41[0], 39)
  ! SEND TEST_POINT_41 MESSAGE TO NETWORK MONITOR !
  ADDRESS_WORD := (((NT02TB[NT02NS] AND %0F) *
    %10) OR (((NT02TB[NT02NS] AND
    %F0) / %10))
  ! MASK OFF RIGHT 4-BITS AND SHIFT RIGHT. MASK OFF
    LEFT 4-BITS AND SHIFT LEFT. THEN COMBINE !
  IF INPUT_SEQ_BIT = TRUE
  THEN CONTROL_WORD := %A0
  ! IF SEQ_BIT_B = 1 THEN CONTROL_WORD = 10100000 !
  ELSE CONTROL_WORD := %80
  ! IF SEQ_BIT_B = 0 THEN CONTROL_WORD = 10000000 !
FI
INDEX := 0
DO
  OUTFRAME_CHB_TB[OUTFRAME_CHB_NE + INDEX] := 0
  INDEX += 1
  IF INDEX > 134 THEN EXIT FI
OD ! CLEAR ALL DATA FOR TRANSMISSION OF S-FRAME !
OUTFRAME_CHB_TB[OUTFRAME_CHB_NE+0] := ADDRESS_WORD
OUTFRAME_CHB_TB[OUTFRAME_CHB_NE+1] := CONTROL_WORD
! MOVE DATA TO OUTGOING TABLE FOR XMISSION OUT CH-B !
LD_TAB_HSKP('04')
! HOUSEKEEP THE OUTFRAME_CHB_TB TABLE !

FI ! END OF IF TABLE CONDITION !

```

[illegible]

INPUT - NONE

OUTPUT - THE OUTPUT IS THE VARIABLE 'TIMCHA'. WHEN TRUE THE TIME DELAY SEQUENCE IS COMPLETE.

NOTES - 1. SEE PROCEDURE STCTC2 FOR AN EXAMPLE OF CALCULATING THE APPROPRIATE TIME DELAY.

INTERNAL

TIME_DELAY_CHA PROCEDURE

ENTRY

```
IF TIMCHA = TRUE
  THEN STCTC2 FI
  ! IF TIMER IS NOT RUNNING THEN START IT !
  IF CTCNOA >= MAXNOA
    THEN
      TIMCHA := TRUE
      CTCNOA := 0
    FI
```

END TIME_DELAY_CHA

!*****!

!*****!

PROCEDURE TIME_DELAY_CHB PRODUCES A DELAY OF TIME FOR CHANNEL-B

THE PURPOSE OF THIS PROCEDURE IS TO PRODUCE A TIME DELAY TO BE USED
BETWEEN SUCCESSIVE TRANSMISSIONS OUT CHANNEL-B OF I-FRAMES WHEN A VALID
ACKNOWLEDGEMENT HAS NOT BEEN RECEIVED.

INPUT - NONE.

PROCESSING - THE PROCEDURE BEGINS BY CALLING THE ASSEMBLY ROUTINE STCTC3
WHICH INITIALIZES AND STARTS THE CTC CHANNEL-3. EACH TIME THE CTC RUNS
OUT, ITS INTERRUPT ROUTINE (TOUTCB) INCREMENTS THE GLOBAL VARIABLE
'CTCNOB'. WHENEVER THE TOUTCB VALUE IS GREATER OR EQUAL TO THE
GLOBAL VARIABLE 'MAXNOB', THE GLOBAL VARIABLE 'TIMCHB' IS CHANGED
TO TRUE.

OUTPUT - THE OUTPUT IS THE VARIABLE 'TIMCHB'. WHEN TRUE THE TIME DELAY

SEQUENCE IS COMPLETE.

INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE ROUTE_OUT LOCATED
WITHIN THIS SAME MODULE. THIS PROCEDURE CALLS PROCEDURE STCTC3 WHICH
IS LOCATED IN THE ASSEMBLY MODULE N.INSIO_U2.

NOTES - 1. SEE PROCEDURE STCTC3 FOR AN EXAMPLE OF CALCULATING THE
APPROPRIATE TIME DELAY.

2. VARIABLES TOUTCB, CTCNOB, AND MAXNOB ALLOW CHANNEL-B TO BE
INDEPENDENT OF CHANNEL-A WHEN CONSIDERING TIME DELAY TO AN
ADJACENT UNID.

INTERNAL
*****!

TIME_DELAY_CHB PROCEDURE

ENTRY

```
IF TIMCHB = TRUE
  THEN STCTC3 FI
  ! IF THE TIMER IS NOT RUNNING THEN START IT !
  IF CTCNOB >= MAXNOB
    THEN
      TIMCHB := TRUE
      CTCNOB := 0
    FI
```

END TIME_DELAY_CHB

*****!

*****!

PROCEDURE ROUTE_IN ROUTES PACKETS IN FROM EITHER CH-A OR CH-B

THE PURPOSE OF THIS PROCEDURE IS TO ROUTE PACKETS FROM

THE NETWORK INPUT TABLES TO THEIR CORRECT OUTPUT TABLE.
IT INITIATES THE BUILDING OF AN S-FRAME FOR ACKNOWLEDGEMENT
OF A GOOD I-FRAME. IT ALSO INSURES THE PROPER SEQUENCING
OF FRAMES BY THE USE OF MODULO 2 NUMBERING SCHEME.

INPUT - DATA PACKETS OR FRAMES ARE ROUTED VIA EVALUATION
OF NT01TB AND NT02TB POINTERS AND FRAME HEADER ROUTING
INFORMATION.

PROCESSING - THE PROCEDURE CHECKS NT01TB AND NT02TB POINTERS FOR
FRAME ARRIVAL. IT THEN DETERMINES IF THE FRAME IS DESTINED
FOR THIS UNID OR ANOTHER UNID. IF FOR ANOTHER UNID, THE PROCEDURE
SIMPLY ROUTES IT BACK TO THE NETWORK. IF THE INCOMING FRAME WAS ON
CH-A, THEN IT IS SUBSEQUENTLY TRANSMITTED OUT CH-B. IF THE INCOMING
FRAME WAS ON CH-B, THEN IT IS ROUTED OUT CH-A. THIS WILL CREATE
A DUAL RING ON THE NETWORK SIDE OF THE UNIDS. ONE DIRECTION THE
FRAMES ARE MOVING CLOCKWISE THE OTHER COUNTER-CLOCKWISE.

IF THE INCOMING FRAME IS DESTINED FOR THIS PARTICULAR UNID THEN
IT DETERMINES THE TYPE OF FRAME. IF IT IS AN I-FRAME THEN
IT CALLS THE BUILD_S_FRAME PROCEDURE TO PROVIDE A POSITIVE
ACKNOWLEDGEMENT AND THEN MOVES THE FRAME TO THE NETWORK-TO-LOCAL
TABLE MINUS THE FRAME HEADER. IF IT IS AN S-FRAME THEN IT TESTS TO
SEE IF IT IS AN ACKNOWLEDGEMENT FOR ITS LAST TRANSMITTED I-FRAME.
THIS PROCEDURE USES PROCEDURE-DET_DEST_ONE AND DET_DEST_TWO TO
DETERMINE DESTINATION AND THE MOVSEQ PROCEDURE TO MOVE THE FRAME
BETWEEN TABLES.

OUTPUT - THIS PROCEDURE PRODUCES TWO OUTPUTS. THE FIRST IS A FRAME
FROM THE NETWORK SIDE DESTINED FOR ONE OF THE OUTGOING LOCAL
CHANNELS. THE SECOND OUTPUT IS A FRAME FROM THE NETWORK DESTINED
FOR ANOTHER UNID LOCATED ON THE NETWORK.

INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY PROCEDURE
MAIN.

NOTES - NONE.

 INTERNAL

ROUTE_IN PROCEDURE

ENTRY

```

IF ((NT01NE - NT01NS) >= FRAME_SIZE)
  ORIF (NT01NS > NT01NE)
  THEN
    SNDSEQ(CONCMD,CONDAT, TP_5[0],50)
    ! SEND TEST_POINT_5 TO NETWORK MONITOR !
    DESTINATION := DET_DEST_ONE('01')
    IF DESTINATION
      CASE 'NN' ! FRAME DESTINED FOR ANOTHER UNID !
      THEN
        IF (NT01TB [NT01NS+1] AND %20) = %20
          ! CHECKS FOR THE SEQUENCE BIT !
          THEN ! SEQ_BIT_A = 1 !
            SNDSEQ(CONCMD,CONDAT, TP_49[0],28)
            ! SEND TEST_POINT_49 MSG TO NETWORK MONITOR !
            INPUT_SEQ_BIT := TRUE
            BUILD_S_FRAME('01',INPUT_SEQ_BIT)
          ELSE ! SEQ_BIT_A = 0 !
            SNDSEQ(CONCMD,CONDAT, TP_50[0],28)
            ! SEND TEST_POINT_50 MSG TO NETWORK MONITOR !
            INPUT_SEQ_BIT := FALSE
            BUILD_S_FRAME('01',INPUT_SEQ_BIT)
          FI
        SNDSEQ(CONCMD,CONDAT, TP_6[0],50)
        ! SEND TEST_POINT_6 MESSAGE TO NETWORK MONITOR !
        MOVSEQ( NT01TB[NT01NS],
          OUTFRAME_CHB_TB[OUTFRAME_CHB_NE],

```

```

FRAME_SIZE)
! MOVE FRAME TO CH-B OUTPUT TABLE !
LD_TAB_HSKP ('04')
! UPDATE CH-B OUTPUT TABLE !

CASE 'NL' ! FRAME IS DESTINED FOR THIS UNID !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_7[0],38)
  ! SEND TEST_POINT_7 MESSAGE TO NETWORK MONITOR !
  IF (NT01TB[NT01NS+1] AND %80) = %80
  ! THIS CHECKS FOR A S-FRAME !
  THEN ! HAVE A S-FRAME !
  SNDSEQ(CONCMD,CONDAT, TP_8[0],32)
  ! SEND TEST_POINT_8 MESSAGE TO NETWORK MONITOR !
  IF (NT01TB[NT01NS+1] AND %20) = %20
  ! CHECKS THE SEQUENCE BIT !
  THEN ! SEQ_BIT_A = 1 !
  SNDSEQ(CONCMD,CONDAT, TP_9[0],27)
  ! SEND TEST_POINT_9 MESSAGE TO NETWORK MONITOR!
  THIS_SEQ_BIT := TRUE
  IF THIS_SEQ_BIT = SEQ_BIT_A
  THEN ACKNOWLEDGE_A := TRUE FI
  ELSE ! SEQ_BIT_A = 0 !
  SNDSEQ(CONCMD,CONDAT, TP_10[0],28)
  ! SEND TEST_POINT_10 MSG TO NETWORK MONITOR !
  THIS_SEQ_BIT := FALSE
  IF THIS_SEQ_BIT = SEQ_BIT_A
  THEN ACKNOWLEDGE_A := TRUE FI
FI
ELSE ! HAVE A I-FRAME !
  SNDSEQ(CONCMD,CONDAT, TP_11[0],33)
  ! SEND TEST_POINT_11 MESSAGE TO NETWORK MONITOR !
  IF (NT01TB[NT01NS+1] AND %20) = %20
  ! CHECKS FOR THE SEQUENCE BIT !
  THEN ! SEQ_BIT_A = 1 !

```



```

SNDSEQ(CONCMD,CONDAT, TP_12[0],28)
! SEND TEST_POINT_12 MSG TO NETWORK MONITOR !
INPUT_SEQ_BIT := TRUE
BUILD_S_FRAME ('01', INPUT_SEQ_BIT)
ELSE ! SEQ_BIT_A = 0 !
SNDSEQ(CONCMD,CONDAT, TP_13[0],28)
! SEND TEST_POINT_13 MSG TO NETWORK MONITOR !
INPUT_SEQ_BIT := FALSE
BUILD_S_FRAME ('01', INPUT_SEQ_BIT)
FI

SNDSEQ(CONCMD,CONDAT, TP_45[0],42)
! SEND TEST_POINT_45 MSG TO NETWORK MONITOR !
MOVSEQ( NT01TB[NT01NS+2], NTLCTB[NTLCNE],
        PACKET_SIZE)
LD_TAB_HSKP ('NL')
FI ! END OF %80 CONDITION !

FI ! END OF IF DESTINATION CONDITION !

SRVC_TAB_HSKP ('01')

FI ! END OF IF (NT01NE - NT01NS) CONDITION !

IF ((NT02NE - NT02NS) >= FRAME_SIZE)
ORIF (NT02NS > NT02NE)
THEN
SNDSEQ(CONCMD,CONDAT, TP_14[0],51)
! SEND TEST_POINT_14 MESSAGE TO NETWORK MONITOR !
DESTINATION := DET_DEST_TWO('02')
IF DESTINATION
CASE 'NN' ! FRAME DESTINED FOR ANOTHER UNID !
THEN
IF (NT02TB [NT02NS+1] AND %20) = %20
! CHECKS FOR THE SEQUENCE BIT !

```

```

THEN ! SEQ_BIT_B = 1 !
  SNDSEQ(CONCMD,CONDAT, TP_51[0],28)
  ! SEND TEST_POINT_51 MSG TO NETWORK MONITOR !
  INPUT_SEQ_BIT := TRUE
  BUILD_S_FRAME('02',INPUT_SEQ_BIT)
ELSE ! SEQ_BIT_B = 1 !
  SNDSEQ(CONCMD,CONDAT, TP_52[0],28)
  ! SEND TEST_POINT_52 MSG TO NETWORK MONITOR !
  INPUT_SEQ_BIT := FALSE
  BUILD_S_FRAME('02',INPUT_SEQ_BIT)
FI

SNDSEQ(CONCMD,CONDAT, TP_15[0],51)
! SEND TEST_POINT_15 MESSAGE TO NETWORK MONITOR !
MOVSEQ( NT02TB[NT02NS],
        OUTFRAME_CHA_TB[OUTFRAME_CHA_NE],
        FRAME_SIZE)
! MOVE FRAME TO CH-A OUTPUT TABLE !
LD_TAB_HSKP ('03')
! UPDATE CH-A OUTPUT TABLE !

CASE 'NL' ! FRAME DESTINED FOR THIS UNID !
THEN
  SNDSEQ(CONCMD,CONDAT, TP_16[0],39)
  ! SEND TEST_POINT_16 MESSAGE TO NETWORK MONITOR !
  IF (NT02TB[NT02NS+1] AND %80) = %80
  ! CHECKS FOR PRESENCE OF S-FRAME !
  THEN ! HAVE A S-FRAME !
    SNDSEQ(CONCMD,CONDAT, TP_17[0],32)
    ! SEND TEST_POINT_17 MESSAGE TO NETWORK MONITOR !
    IF (NT02TB [NT02NS+1] AND %20) = %20
    ! CHECKS THE SEQUENCE BIT !
    THEN ! SEQ_BIT_B = 1 !
      SNDSEQ(CONCMD,CONDAT, TP_18[0],28)
      ! SEND TEST_POINT_18 MSG TO NETWORK MONITOR !
      THIS_SEQ_BIT := TRUE

```

```

IF THIS_SEQ_BIT = SEQ_BIT_B
  THEN ACKNOWLEDGE_B := TRUE FI
ELSE ! SEQ_BIT_B = 0 !
  SNDSEQ(CONCMD,CONDAT, TP_19[0],28)
  ! SEND TEST_POINT_19 MSG TO NETWORK MONITOR !
  THIS_SEQ_BIT := FALSE
  IF THIS_SEQ_BIT = SEQ_BIT_B
    THEN ACKNOWLEDGE_B := TRUE FI
FI
ELSE ! HAVE A I-FRAME !
  SNDSEQ(CONCMD,CONDAT, TP_20[0],33)
  ! SEND TEST_POINT_20 MSG TO NETWORK MONITOR !
  IF (NT02TB[NT02NS+1] AND %20) = %20
    ! CHECKS THE SEQUENCE BIT !
    THEN ! SEQ_BIT_B = 1 !
      SNDSEQ(CONCMD,CONDAT, TP_21[0],28)
      ! SEND TEST_POINT_21 MSG TO NETWORK MONITOR !
      INPUT_SEQ_BIT := TRUE
      BUILD_S_FRAME ('02', INPUT_SEQ_BIT)
    ELSE ! SEQ_BIT_B = 0 !
      SNDSEQ(CONCMD,CONDAT, TP_22[0],28)
      ! SEND TEST_POINT_22 MSG TO NETWORK MONITOR !
      INPUT_SEQ_BIT := FALSE
      BUILD_S_FRAME ('02', INPUT_SEQ_BIT)
    FI
  SNDSEQ(CONCMD,CONDAT, TP_46[0],42)
  ! SEND TEST_POINT_46 MSG TO NETWORK MONITOR !
  MOVSEQ( NT02TB[NT02NS+2], NTLCTB[NTLCNE],
    PACKET_SIZE)
  LD_TAB_HSKP ('NL')
  FI ! END OF %80 CONDITION !
FI ! END OF IF DESTINATION CONDITION !

SRVC_TAB_HSKP ('02')

```


THE PURPOSE OF THIS PROCEDURE IS TO ROUTE FRAMES FROM THE LOCAL-TO-NETWORK TABLE (LCNTTB) TO ONE OF THE OUTGOING NETWORK PORTS (OUTFRAME_CHA_TB OR OUTFRAME_CHB_TB) AND TO ROUTE FRAMES FROM THE NETWORK SIDE BACK ONTO THE NETWORK. IT ALSO MAINTAINS FLOW CONTROL AND ERROR RECOVERY BY WAITING FOR THE ROUTE_IN PROCEDURE TO RECEIVE POSITIVE ACKNOWLEDGEMENT FOR I-FRAMES TRANSMITTED BY THIS PROCEDURE.

INPUT - DATA FRAMES ARE ROUTED VIA EVALUATION OF OUTFRAME_CHA_TB, OUTFRAME_CHB_TB, AND LCNTTB POINTERS WITH FRAME HEADER ADDRESS INFORMATION.

PROCESSING - THE PROCEDURE CHECKS EACH INPUT TABLE'S POINTERS FOR FRAMES TO BE TRANSMITTED ONTO THE NETWORK. BEFORE ANY TRANSMISSION OCCURS, MAX_NETWORK_CODE IS COMPARED AGAINST THE DESTINATION ADDRESS TO INSURE A FRAME CANNOT CONTINUE ON THE NETWORK WHEN IT HAS AN ADDRESS GREATER THAN THE AVAILABLE NUMBER OF UNIDS. NET-TO-NET FRAMES ARE SIMPLY TRANSFERRED TO THE NETWORK. ALL FRAMES WHICH ORIGINATE AT THIS UNID ARE CHECKED TO IDENTIFY THE TYPE. S-FRAMES ARE SIMPLY TRANSMITTED. I-FRAMES ARE TRANSMITTED AND ENTER A TIME DELAY WAITING FOR A GOOD ACKNOWLEDGEMENT. IF A GOOD ACKNOWLEDGEMENT IS NOT RECEIVED BY THE END OF THE WAIT PERIOD, IT IS TRANSMITTED AGAIN UPON THE NEXT CYCLE THROUGH THIS PROCEDURE.

OUTPUT - A FRAME OF DATA IS TRANSMITTED TO ONE OF THE NETWORK CHANNELS.

INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE MAIN PROCEDURE.

NOTES - 1. THERE ARE TWO VERY IMPORTANT PARAMETERS RELATING TO THIS PROCEDURE THAT MUST BE CONSIDERED PRIOR TO THE MODIFICATION OF THIS PROCEDURE. THE FIRST IS VARIABLE MAX_NETWORK_CODE. ALL ACTIVE UNIDS ON THE NETWORK WILL START WITH 0 AND PROCEED TO INCREMENT IN A CLOCKWISE MANNER. IN THIS WAY THE ADDRESS CAN

THE SECOND PARAMETER IS MAXNUM. MAXNUM IS THE MAXIMUM NUMBER OF TIMES THE TIMEOUT CYCLE LOOPS THROUGH THE RETRANSMISSION PERIOD FOR AN I-FRAME. THIS PERIOD ALLOWS TIME FOR A VALID ACKNOWLEDGEMENT TO BE RECEIVED AFTER THE I-FRAME IS TRANSMITTED. THE BASIC PERIOD IS SET THROUGH N.INSIO_U2 FOR 27 MILLISECONDS. IF MAXNUM=10 THEN THE WAIT LOOP IS 270 MILLISECONDS.

INTERNAL

ENTRY

F-173

```

        FRAME_SIZE)
    ACKNOWLEDGE_A := FALSE
    TIME_DELAY_CHA
FI
IF ACKNOWLEDGE_A = TRUE
    THEN
        SRVC_TAB_HSKP ('03')
        SEQ_BIT_A := NOT SEQ_BIT_A
FI
ELSE
    SNDSEQ(CONCMD,CONDAT, TP_48[0],50)
    ! SEND TEST_POINT_48 MESSAGE TO NETWORK MONITOR !
    STATTB [11] += 1
    STATTB [14] += 1
    SRVC_TAB_HSKP ('03')
FI
FI ! END (OUTFRAME_CHA_NE-OUTFRAME_CHA_NS) CONDITION !

IF ((OUTFRAME_CHB_NE - OUTFRAME_CHB_NS) >= FRAME_SIZE)
    ORIF (OUTFRAME_CHB_NS > OUTFRAME_CHB_NE)
    THEN
        SNDSEQ(CONCMD,CONDAT, TP_28[0],51)
        IF ((OUTFRAME_CHB_TB[OUTFRAME_CHB_NS] AND %F0)/%10)
            <= MAX_NETWORK_CODE
        THEN
            IF TIMCHB = FALSE
            THEN TIME_DELAY_CHB
        FI
        IF (ACKNOWLEDGE_B = FALSE) ANDIF (TIMCHB = TRUE)
        THEN
            SNDSEQ(CONCMD,CONDAT, TP_29[0],54)
            OUTFRAME_CHB_TB[OUTFRAME_CHB_NS] :=
            OUTFRAME_CHB_TB[OUTFRAME_CHB_NS] AND %F0
            OUTFRAME_CHB_TB[OUTFRAME_CHB_NS] :=
            OUTFRAME_CHB_TB[OUTFRAME_CHB_NS] OR (THIS_UNID_NBR)

```

```

XMITCB( OUTFRAME_CHB_TB[OUTFRAME_CHB_NS],
        FRAME_SIZE)
ACKNOWLEDGE_B := FALSE
TIME_DELAY_CHB
FI
IF ACKNOWLEDGE_B = TRUE
THEN
    SRVC_TAB_HSKP ('04')
    SEQ_BIT_B := NOT SEQ_BIT_B
FI
ELSE
    SNDSEQ(CONCMD,CONDAT, TP_47[0],50)
    ! SEND TEST_POINT_47 MESSAGE TO NETWORK MONITOR !
    STATTB[11] += 1
    STATTB[14] += 1
    SRVC_TAB_HSKP ('04')
FI
FI ! END (OUTFRAME_CHB_NE - OUTFRAME_CHB_NS) CONDITION !

END ROUTE_OUT

!*****!

GLOBAL

!*****!
PROCEDURE
MAIN
    PROCEDURE FOR MAIN LINE DRIVER OF NETWORK OS

    THE PURPOSE OF THIS PROCEDURE IS TO PROVIDE THE MAIN LINE
    OF PROCESSING FOR N.OS.

INPUT - NONE.

PROCESSING - THIS PROCEDURE SENDS A HEADER TO THE NETWORK MONITOR
CONSOLE, INITIALIZES THE NETWORK TABLES VIA INIT_N_TAB,

```


USES INSIO TO INITIALIZE THE SIO INTERRUPTS,
AND LOOPS ENDLESSLY ROUTING FRAMES IN AND OUT VIA PROCEDURES
ROUTE_IN AND ROUTE_OUT.

OUTPUT - A START UP MESSAGE IS SENT TO THE NETWORK MONITOR UPON START UP.

INTERFACE - THIS PROCEDURE IS THE INITIAL ENTRY POINT FOR N.OS.
IT OPERATES THROUGH SINGLE CALLS TO SNDSEQ, INIT_N_TAB, AND
INSIO; AND REPETITIVE CALLS TO ROUTE_IN AND ROUTE_OUT.

NOTES - NONE.

MAIN PROCEDURE

ENTRY

```

SNDSEQ(CONCMD, CONDAT, STARTUP_HDR[0], 51)
! SEND START-UP HEADER TO NETWORK MONITOR !

SNDSEQ(CONCMD, CONDAT, TP_1[0], 37)
! SEND TEST_POINT_1 MESSAGE TO NETWORK MONITOR !

INIT_N_TAB

SNDSEQ(CONCMD, CONDAT, TP_2[0], 32)
! SEND TEST_POINT_2 MESSAGE TO NETWORK MONITOR !

INSIO

SNDSEQ(CONCMD, CONDAT, TP_3[0], 40)
! SEND TEST_POINT_3 MESSAGE TO NETWORK MONITOR !

DO

```

SNDSEQ(CONCMD,CONDAT, TP_4[0],35)
! SEND TEST_POINT_4 MESSAGE TO NETWORK MONITOR !

ROUTE_IN

SNDSEQ(CONCMD,CONDAT, TP_42[0],37)
! SEND TEST_POINT_42 MESSAGE TO NETWORK MONITOR !

ROUTE_OUT

SNDSEQ(CONCMD,CONDAT, TP_43[0],39)
! SEND TEST_POINT_43 MESSAGE TO NETWORK MONITOR !

OD

SNDSEQ(CONCMD,CONDAT, TP_44[0],44)
! SEND TEST_POINT_44 MESSAGE TO NETWORK MONITOR !

END MAIN

END MAIN

!*****!

!*****!

```

!*****
MODULE  N_TAB_U2      NETWORK OS TABLES      DATE ORIGINATED: 9 NOV 82
                                           DATE LAST MODIFIED: _____

      THE PURPOSE OF THIS MOODULE IS TOO PROVIDE THE NETWORK
      OPERATING SYSTEM (N.OS) WITH THE TABLES REQUIRED FOR DATA PROCESSING.
      THIS MODULE CONSISTS PRIMARILY OF TABLE DEFINITIONS WITH PROCESSING
      LIMITED TO THE INITILIZATION OF THE DEFINED TABLES VIA INIT_N_TAB.

      THE TABLE SIZE IS A FUNCTION OF FRAME SIZE RATHER
      THAN PACKET SIZE. THIS IS DUE TO THESE TABLES OPERATING WITHIN
      THE NETWORK SIDE OF THE UNID. CURRENTLY, MINIMAL X.25
      FRAMING IS BEING ACCOMPLISHED WITH A THREE BYTE HEADER AND
      A THREE BYTE TRAILER. THE Z80-SIO STRIPS OFF THE TWO FLAG BYTES
      AND THE TWO BYTES USED FOR ERROR CHECKING.
*****
N_TAB_U2 MODULE

```

```

CONSTANT
  PACKET_SIZE := 133
  PACKETS_IN_TABLE := 10
  FRAME_SIZE := PACKET_SIZE + 2  ! 2-BYTES FOR FRAME HEADER !
  FRAMES_IN_TABLE := 10
  FRAME_TABLE_SIZE := FRAME_SIZE * FRAMES_IN_TABLE

```

```

GLOBAL

NT01TB ARRAY [FRAME_TABLE_SIZE BYTE]
NT01NS INTEGER
NT01NE INTEGER
NT01SZ INTEGER

NT02TB ARRAY [FRAME_TABLE_SIZE BYTE]
NT02NS INTEGER
NT02NE INTEGER
NT02SZ INTEGER

```

```

OUTFRAME_CHA_TB ARRAY [FRAME_TABLE_SIZE BYTE]
OUTFRAME_CHA_NS INTEGER
OUTFRAME_CHA_NE INTEGER
OUTFRAME_CHA_SZ INTEGER

```

```

OUTFRAME_CHB_TB ARRAY [FRAME_TABLE_SIZE BYTE]
OUTFRAME_CHB_NS INTEGER
OUTFRAME_CHB_NE INTEGER
OUTFRAME_CHB_SZ INTEGER

```

```

!*****
PROCEDURE INIT_N_TAB PROCEDURE TO INITIALIZE NETWORK DATA TABLES
*****

```

THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE
DATA TABLES USED BY THE UNIDS NETWORK OPERATING SYSTEM.

```

INPUT - NONE.

```

```

PROCESSING - THE PROCESS INITIALIZES BOTH NETWORK CHANNEL
INPUTS AND NETWORK-TO-NETWORK TABLES BY SETTING
THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-
BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO
A MULTIPLE OF FRAME_SIZE.

```

```

OUTPUT - THE TABLE POINTERS AND STATUS TABLE AS NOTED UNDER
PROCESSING ARE MODIFIED.

```

```

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-MAIN.

```

```

NOTES - NONE.

```

```

*****
INIT_N_TAB PROCEDURE
*****!

```

```

ENTRY

```

```

! INITIALIZE MEMORY TABLE INFORMATION
  xxxxns - NEXT BYTE TO BE SERVICED
  xxxxne - NEXT EMPTY BYTE
  xxxxsZ - SIZE OF DATA TABLE
!

! INITIALIZE NETWORK CHANNEL-1 INPUT TABLE !

NT01ns := 0
NT01ne := 0
NT01sZ := FRAME_TABLE_SIZE

! INITIALIZE NETWORK CHANNEL-2 INPUT TABLE !

NT02ns := 0
NT02ne := 0
NT02sZ := FRAME_TABLE_SIZE

! INITIALIZE OUTGOING NETWORK TABLE FOR CH-A !

OUTFRAME_CHA_ns := 0
OUTFRAME_CHA_ne := 0
OUTFRAME_CHA_sZ := FRAME_TABLE_SIZE

! INITIALIZE OUTGOING NETWORK TABLE FOR CH-B !

OUTFRAME_CHB_ns := 0
OUTFRAME_CHB_ne := 0
OUTFRAME_CHB_sZ := FRAME_TABLE_SIZE

```

END INIT_N_TAB

!*****!

END N_TAB_U2

!*****!

!*****!

Appendix F Section V

This section of Appendix F contains the software listings which comprise the DELNET Monitor (Zilog MCZ 1/25 Computer System) operating system.

```

!*****
PROLOGUE -      MODULE ZILOG_APPLICATION      DATE ORIGINATED: 29 JUL 83
                                         DATE LAST MODIFIED: _____

      THE PURPOSE OF THIS MODULE IS TO PROVIDE THE ZILOG_HOST
      WITH THE APPLICATION LAYER PROTOCOL. THE ZILOG_HOST IS REQUIRED TO
      INTERFACE WITH THE UNID FOR BOTH SENDING AND RECEIVING DATA.

      THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE-MAIN, AND
      SUBORDINATE PROCEDURES TRNMIT_DATA, SRVC_TAB_HSKP, LD_TAB_HSKP,
      DET_DEST_ONE, DET_DEST_TWO, LOAD_OUTGOING_DATA, ROUTE_IN,
      AND LASTLY ROUTE_OUT.
!*****
MAIN MODULE

```

TYPE

PBYTE ^BYTE

CONSTANT

```

! FOLLOWING VALUES ARE PARTICULAR TO THE ZILOG MCZ 1/25 SYSTEM !
PRINTER_DATA_PORT := %90      ! PRINTER DATA PORT ADDRESS !
PRINTER_COMMAND_PORT := %91    ! PRINTER COMMAND PORT ADDRESS !
H19_COMMAND_PORT := %DF        ! H19 CONSOLE COMMAND PORT ADDRESS !
H19_DATA_PORT := %DE           ! H19 CONSOLE DATA PORT ADDRESS !

DATA_SIZE := 128               ! IN/OUT DATA IN 128-BYTE BLOCKS !
DATA_TABLE := 25               ! WILL BUFFER 25 DATAGRAMS !
DATA_TABLE_SIZE := DATA_SIZE * DATA_TABLE
ERROR_TABLE_SIZE := 128        ! WILL ALLOW 128 DIFFERENT ERROR MSGS !
J112_DATA_PORT := %8C          ! ZILOG J-112 PORT DATA ADDRESS !
J113_DATA_PORT := %8E          ! ZILOG J-113 PORT DATA ADDRESS !

EXTERNAL      ! LOCATED IN ZILOG_TRANSPORT !

```

```
BUILD_TRANSPORT_HEADER PROCEDURE
EXTRACT_TRANSPORT_HEADER PROCEDURE
```

```
EXTERNAL      ! LOCATED IN ZILOG_VECINT !
```

```
VECINT PROCEDURE
```

```
EXTERNAL      ! LOCATED IN ZILOG_LIB !
```

```
SNDSEQ PROCEDURE (CMDPRT BYTE,DATPRT BYTE,BYTADD PBYTE,NUMBYT BYTE)
RECSEQ PROCEDURE (CMDPRT BYTE,DATPRT BYTE,BYTADD PBYTE,NUMBYT BYTE)
MOVSEQ PROCEDURE (SRCADD PBYTE,DTDADD PBYTE,NUMBYT BYTE)
TRNMIT PROCEDURE
```

```
EXTERNAL      ! LOCATED IN ZILOG_TABLES !
```

```
INIT_ZILOG_TABLES PROCEDURE
```

```
ZL01TB ARRAY [DATA_TABLE_SIZE BYTE]
ZL01NS INTEGER
ZL01NE INTEGER
ZL01SZ INTEGER
```

```
ZL02TB ARRAY [DATA_TABLE_SIZE BYTE]
ZL02NS INTEGER
ZL02NE INTEGER
ZL02SZ INTEGER
```

```
ERROR_TABLE ARRAY [ERROR_TABLE_SIZE BYTE]
```

```
INCOMING_DATA_BLOCK ARRAY [DATA_SIZE BYTE]
OUTGOING_DATA_BLOCK ARRAY [DATA_SIZE BYTE]
```

```
GLOBAL
```



```

DATADD      PBYTE      ! CHANNEL TRANSMIT DATA ADDRESS !
OUTADD      BYTE       ! CHANNEL TRANSMIT PORT ADDRESS !
DATA_PRESENT INTEGER

```

```

! SOURCE AND DESTINATION ADDRESS'S ARE ONLY PRESET VALUES !
SOURCE_ADDRESS_ONE BYTE
SOURCE_ADDRESS_TWO BYTE
SOURCE_ADDRESS_THREE BYTE
SOURCE_ADDRESS_FOUR BYTE
DESTINATION_ADDRESS_ONE BYTE
DESTINATION_ADDRESS_TWO BYTE
DESTINATION_ADDRESS_THREE BYTE
DESTINATION_ADDRESS_FOUR BYTE

```

```

INTERNAL ! INTERNAL VARIABLES USED IN THIS MODULE !

```

```

DESTINATION WORD
PORT_ADDRESS BYTE
LENGTH WORD
RCODE BYTE

```

```

TEST_MSG_1 ARRAY [*BYTE] := 'NOW IS THE TIME FOR ALL GOOD MEN TO'
                                ' COME TO THE AID OF THEIR COUNTRY.'

```

```

TEST_MSG_2 ARRAY [*BYTE] :=

```

```

'The Gettysburg Address by President Abraham Lincoln: %R%L'
'%R%L'

```

```

Four score and seven years ago our fathers brought forth %R%L'
on this continent, a new nation, conceived in Liberty, and dedicated %R%L'
to the proposition that all men are created equal. %R%L'

```

```

Now we are engaged in a great civil war, testing whether %R%L'
that nation, or any other nation so conceived and so dedicated can %R%L'
long endure. We are met on a great battlefield of that war. We have %R%L'
come to dedicate a portion of that field, as a final resting place for %R%L'
those who here gave their lives that that nation might live. It is %R%L'

```

'altogether fitting and proper that we should do this. But, in a larger %R&L'
'sense, we cannot dedicate-we cannot consecrate-we cannot hallow-this %R&L'
'ground. %R&L'

The brave men, living and dead, who struggled here, have %R&L'
'consecrated it, far above our poor power to add or detract. The world %R&L'
'will little note, nor long remember what we say here, but it can never %R&L'
'forget what they did here. It is for us the living, rather, to be %R&L'
'dedicated here to the unfinished work which they who fought here have %R&L'
'thus far nobly advanced. It is rather for us to be here dedicated to the %R&L'
'great task remaining before us-that from these honored dead we take %R&L'
'increased devotion to that cause for which they gave the last full %R&L'
'measure of devotion-that we here highly resolve that these dead shall %R&L'
'not have died in vain-that this nation, under God, shall have a new %R&L'
'birth of freedom-and that government of the people, by the people, %R&L'
'for the people, shall not perish from the Earth. %R&L'

! TEST POINTS TO AID IN DEBUGGING !

STARTUP_HDR ARRAY [*BYTE] := '%R&L'
'ZILOG HOST OS%R&L'
'VERSION 15 SEP 83%R&L'
'EXECUTING%R&L'

TP_1	ARRAY	[*BYTE]	:=	'%R<P_1:	INCOMING	ZL01TB	CONTROL	CODE	VALID'
TP_2	ARRAY	[*BYTE]	:=	'%R<P_2:	INCOMING	ZL01TB	CONTROL	CODE	INVALID'
TP_3	ARRAY	[*BYTE]	:=	'%R<P_3:	INCOMING	ZL02TB	CONTROL	CODE	VALID'
TP_4	ARRAY	[*BYTE]	:=	'%R<P_4:	INCOMING	ZL02TB	CONTROL	CODE	INVALID'
TP_63	ARRAY	[*BYTE]	:=	'%R<P_63:	INCOMING	DATA	LOCATED	IN	ZL01TB'
TP_64	ARRAY	[*BYTE]	:=	'%R<P_64:	INCOMING	DATA	GOING	TO	PRINTER'
TP_65	ARRAY	[*BYTE]	:=	'%R<P_65:	INCOMING	DATA	GOING	TO	H19 TERMINAL'
TP_66	ARRAY	[*BYTE]	:=	'%R<P_66:	INCOMING	DATA	LOCATED	IN	ZL02TB'
TP_67	ARRAY	[*BYTE]	:=	'%R<P_67:	INCOMING	DATA	GOING	TO	PRINTER'
TP_68	ARRAY	[*BYTE]	:=	'%R<P_68:	INCOMING	DATA	GOING	TO	H19 TERMINAL'
TP_69	ARRAY	[*BYTE]	:=	'%R<P_69:	SETTING	OUTGOING	DATA	BLOCK	TO ZEROS'
TP_70	ARRAY	[*BYTE]	:=	'%R<P_70:	DETERMINE	DATAGRAM	SOURCE	ADDRESS'	

```

TP_71 ARRAY [*BYTE] := '%R&LTP_71: DETERMINE DATAGRAM DESTINATION ADDRESS'
TP_72 ARRAY [*BYTE] := '%R&LTP_72: ENTER BUILD TRANSPORT HEADER PROCEDURE'
TP_73 ARRAY [*BYTE] := '%R&LTP_73: ENTER LOAD OUTGOING DATA PROCEDURE'
TP_74 ARRAY [*BYTE] := '%R&LTP_74: TRANSMIT THE DATA TO THE UNID'
TP_75 ARRAY [*BYTE] := '%R&LTP_75: ENTERING VECINT PROCEDURE'
TP_76 ARRAY [*BYTE] := '%R&LTP_76: ENTERING INIT ZILOG TABLES PROCEDURE'
TP_77 ARRAY [*BYTE] := '%R&LTP_77: ENTERING ROUTE OUT PROCEDURE'
TP_78 ARRAY [*BYTE] := '%R&LTP_78: ENTERING ROUTE IN PROCEDURE'
TP_79 ARRAY [*BYTE] := '%R&LTP_79: END OF PROGRAM'
TP_80 ARRAY [*BYTE] := '%R&L'

```

```

!*****
PROCEDURE TRANSMIT_DATA SENDS 128-BYTE DATA BLOCK OUT AN IDENTIFIED PORT

```

THE PURPOSE OF THIS PROCEDURE IS TO SEND A 128-BYTE DATA BLOCK OUT AND IDENTIFIED PORT.

INPUT - THIS PROCEDURE EXPECTS TWO INPUTS: THE FIRST IS THE OUTGOING PORT ADDRESS AND THE SECOND IS THE OUTGOING DATA ADDRESS OF THE DESIRED PORT.

PROCESSING - THE PROCEDURE WILL SEQUENTLY SEND OUT 128-BYTES OF DATA OUT THE IDENTIFIED PORT THRU THE ASSEMBLY LANGUAGE PROGRAM-TRNMIT.
AFTER 128-BYTES HAVE BEEN SENT THE PROCEDURE RETURNS TO THE CALLING MODULE.

OUTPUT - THE OUTPUT IS 128-BYTES SENT OUT AN IDENTIFIED PORT.

INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE-MAIN.

NOTES - NONE.

```

*****
INTERNAL
*****!

```

TRANSMIT_DATA PROCEDURE (SRCADD PBYTE, PRTADD BYTE, BYTES_TO_SEND INTEGER)

LOCAL

IX INTEGER

ENTRY

```
IX := 0
DATADD := SRCADD      ! LOAD DATA ADDRESS FOR TRANSMIT !
OUTADD := PRTADD      ! LOAD PORT ADDRESS FOR TRANSMIT !
DO
  TRNMIT
  IX += 1
  DATADD := INC DATADD
  IF IX = BYTES_TO_SEND THEN EXIT FI
OD
```

END TRANSMIT_DATA

```
!*****!
!*****!
PROCEDURE SRVC_TAB_HSKP      SERVICE TABLE HOUSEKEEP
```

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED TABLE
AFTER SERVICING A 128-BYTE DATA BLOCK.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
REQUIRING HOUSEKEEPING.

PROCESSING - THIS PROCEDURE DETERMINES THE TABLE TO BE PROCESSED, ADVANCES
THE NEXT-BYTE-TO-BE-SERVICED POINTER BY A DATA_SIZE, AND ADJUSTS
THE TABLE FOR WRAP-AROUND IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-BYTE-TO-BE-SERVED POINTER
ADVANCED BY A DATA_SIZE.
INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN AND ROUTE_OUT.

NOTES: NONE.

INTERNAL

SRVC_TAB_HSKP PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

```
CASE '01' ! HOUSEKEEP INCOMING TABLE-1 !
THEN
  ZL01NS := ZL01NS + DATA_SIZE
  IF ZL01NS >= ZL01SZ
  THEN
    ZL01NS := 0 ! ADJUST FOR WRAP-AROUND !
  FI

CASE '02' ! HOUSEKEEP INCOMING TABLE-2 !
THEN
  ZL02NS := ZL02NS + DATA_SIZE
  IF ZL02NS >= ZL02SZ
  THEN
    ZL02NS := 0 ! ADJUST FOR WRAP-AROUND !
  FI
```

FI ! END OF IF TABLE CONDITION !

END SRVC_TAB_HSKP

!*****! !

!*****! !

PROCEDURE LD_TAB_HSKP

LOAD TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED TABLE
AFTER THE LOADING OF A 128-BYTE DATA BLOCK.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
REQUIRING HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED, ADVANCES
THE NEXT-EMPTY-BYTE POINTER BY A DATA_SIZE, AND ADJUSTS FOR TABLE
WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT-EMPTY-BYTE POINTER ADVANCED
BY ONE DATA_SIZE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

NOTES: NONE.

*****! !

INTERNAL

LD_TAB_HSKP PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE '01' ! HOUSEKEEP INCOMING TABLE-1 !
THEN
ZL01NE := ZL01NE + DATA_SIZE

```

IF ZL01NE >= ZL01SZ
THEN
  ZL01NE := 0 ! ADJUST FOR WRAP-AROUND !
FI

CASE '02' ! HOUSEKEEP INCOMING TABLE-2 !
THEN
  ZL02NE := ZL02NE + DATA_SIZE
  IF ZL02NE >= ZL02SZ
  THEN
    ZL02NE := 0 ! ADJUST FOR WRAP-AROUND !
  FI

  FI ! END OF IF TABLE CONDITION !

END LD_TAB_HSKP

!*****!
!*****!
PROCEDURE DET_DEST DETERMINES THE DESTINATION OF DATA FROM THE UNID

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION
OF INCOMING 128-BYTE DATA BLOCKS FROM THE UNID.

INPUT - NONE.

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL_CODE FROM THE INCOMING
DATA TO DETERMINE WHICH ADDRESSING SCHEME IS USED. THE OUTGOING PORT
ADDRESS IS THEN EXTRACTED. THE HEXIDECIMAL VALUE OF THE PORT ADDRESS IS
INSERTED INTO PORT_ADDRESS WITH A RETURN TO THE CALLING MODULE.

OUTPUT - THIS PROCEDURE RETURNS THE PORT_ADDRESS VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_IN.

NOTES: 1. CONTROL_CODE = 00 IS THE ONLY CODE CURRENTLY IMPLEMENTED.

INTERNAL
*****!

DET_DEST PROCEDURE
RETURNS (DESTINATION WORD, PORT_ADDRESS BYTE)

LOCAL

CONTROL_CODE BYTE
PORT_CODE_HIGH BYTE
PORT_CODE_LOW BYTE

ENTRY

CONTROL_CODE := DESTINATION_ADDRESS_ONE AND %F0
IF CONTROL_CODE = 00
THEN
DESTINATION := 'OK'
!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_1[0], 42) !
PORT_CODE_HIGH := DESTINATION_ADDRESS_THREE AND %F0
PORT_CODE_LOW := DESTINATION_ADDRESS_FOUR AND %F0
PORT_ADDRESS := (((PORT_CODE_HIGH) * %10))
OR (((PORT_CODE_LOW) / %10))
ELSE
DESTINATION := 'ER'
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_2[0], 44)
ERROR_TABLE [00] += 1 ! UPDATE ROUTE_IN ERROR COUNT !
ERROR_TABLE [02] += 1 ! INCREMENT CONTROL_CODE ERROR !
FI ! END IF CONTROL_CODE CONDITION !
END DET_DEST


```

!*****!
!*****
PROCEDURE  LOAD_OUTGOING_DATA  LOADS THE OUTGOING DATAGRAM WITH DATA

        THE PURPOSE OF THIS PROCEDURE IS TO LOAD THE REMAINDER OF THE
        DATAGRAM WITH THE OUTGOING DATA.

INPUT ~ THE INPUT TO THIS PROCEDURE IS THE OUTGOING DATA.

PROCESSING - THE PROCEDURE CONTINUES TO INSERT DATA UNTIL EITHER THERE IS
              NO MORE DATA OR THE END OF THE DATAGRAM IS REACHED (128-BYTES MAX).

OUTPUT - FULLY FILLED DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ROUTE_OUT.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST LOADS IN A TEST MSG.
*****!

```

INTERNAL

```

LOAD_OUTGOING_DATA PROCEDURE (DATAGRAM_LENGTH INTEGER,
                              TEST_MSG_LENGTH INTEGER,
                              TEST_MSG_START INTEGER)

```

LOCAL

```

        IX INTEGER
        IY INTEGER
        DATA_LENGTH INTEGER

```

ENTRY

*****!
INTERNAL

ROUTE_IN PROCEDURE

LOCAL

PORT_ADDRESS BYTE
DATAGRAM_LENGTH INTEGER
BYTES_TO_SEND INTEGER

ENTRY

DATAGRAM_LENGTH := 56
IF ((ZL01NE-ZL01NS) >= DATA_SIZE)
ORIF (ZL01NS > ZL01NE)
THEN
DATA_PRESENT := 1
BYTES_TO_SEND := DATA_SIZE - DATAGRAM_LENGTH
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_63[0], 40)
MOVSEQ(ZL01TB[ZL01NS], INCOMING_DATA_BLOCK[0], DATA_SIZE)
EXTRACT_TRANSPORT_HEADER
DESTINATION, PORT_ADDRESS := DET_DEST
IF PORT_ADDRESS = PRINTER_DATA_PORT
THEN
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_64[0], 39)
TRANSMIT_DATA(INCOMING_DATA_BLOCK[DATAGRAM_LENGTH],
PRINTER_DATA_PORT, BYTES_TO_SEND)
FI
IF PORT_ADDRESS = H19_DATA_PORT
THEN
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_65[0], 44)
TRANSMIT_DATA(INCOMING_DATA_BLOCK[DATAGRAM_LENGTH],
H19_DATA_PORT, BYTES_TO_SEND)

```

FI
  SRVC_TAB_HSKP('01')
FI

IF ((ZL02NE-ZL02NS) >= DATA_SIZE)
ORIF (ZL02NS > ZL02NE)
THEN
  DATA_PRESENT := 1
  BYTES_TO_SEND := DATA_SIZE - DATAGRAM_LENGTH
  SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_66[0], 40)
  MOVSEQ(ZL02TB[ZL02NS], INCOMING_DATA_BLOCK[0], DATA_SIZE)
  EXTRACT_TRANSPORT_HEADER
  DESTINATION, PORT_ADDRESS := DET_DEST
  IF PORT_ADDRESS = PRINTER_DATA_PORT
  THEN
    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_67[0], 39)
    TRANSMIT_DATA( INCOMING_DATA_BLOCK[DATAGRAM_LENGTH],
                  PRINTER_DATA_PORT, BYTES_TO_SEND)
  FI
  IF PORT_ADDRESS = H19_DATA_PORT
  THEN
    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_68[0], 44)
    TRANSMIT_DATA( INCOMING_DATA_BLOCK[DATAGRAM_LENGTH],
                  H19_DATA_PORT, BYTES_TO_SEND)
  FI
  SRVC_TAB_HSKP('02')
FI

END ROUTE_IN

!*****!
!*****!
PROCEDURE ROUTE_OUT
  CREATES AND TRANSMITS A 128-BYTE DATAGRAM

```

THE PURPOSE OF THIS PROCEDURE IS TO CREATE A DATAGRAM AND TRANSMIT IT TO THE UNID THAT IS SERVICING THIS HOST.

INPUT - NONE.

PROCESSING - THIS PROCEDURE CREATES A 128-BYTE DATAGRAM BY FIRST ZEROING THE OUTGOING TABLE AND THEN BUILDING A DUMMY TRANSPORT HEADER. THE DATAGRAM IS THEN TRNASMITTED TO THE UNID.

OUTPUT - A 128-BYTE DATAGRAM TO THE UNID.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-MAIN.

NOTES: 1. CURRENTLY THIS PROCEDURE JUST TRANSMITS A DUMMY TRANSPORT HEADER PLUS A TEST MESSAGE TO THE UNID.

*****!

INTERNAL

ROUTE_OUT PROCEDURE

LOCAL

IX INTEGER
DATA_POINTER INTEGER
DATAGRAM_LENGTH INTEGER
TEST_MSG_1_LENGTH INTEGER
TEST_MSG_2_LENGTH INTEGER
TEST_MSG_START INTEGER
TEST_MSG_LENGTH INTEGER
BYTES_TO_SEND INTEGER

ENTRY

DATA_POINTER := 0

```

DATAGRAM_LENGTH := 56
TEST_MSG_1_LENGTH := 69
TEST_MSG_2_LENGTH := 1584
TEST_MSG_START := 0
  BYTES_TO_SEND := DATA_SIZE
DO 1 START AND SEND OUT DATAGRAMS !
  IX := 0
  SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_69[0], 45)
DO
  OUTGOING_DATA_BLOCK[IX] := 0
  IX += 1
  IF IX = BYTES_TO_SEND THEN EXIT FI
OD

SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_70[0], 42)
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_71[0], 47)
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_72[0], 47)
BUILD_TRANSPORT_HEADER
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_73[0], 43)
TEST_MSG_LENGTH := TEST_MSG_2_LENGTH
LOAD_OUTGOING_DATA (DATAGRAM_LENGTH, TEST_MSG_LENGTH,
  TEST_MSG_START)
TEST_MSG_START := TEST_MSG_START + (128-DATAGRAM_LENGTH)
DATA_POINTER := DATA_POINTER + (128-DATAGRAM_LENGTH)
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_74[0], 38)
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_80[0], 2)
SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT,
  OUTGOING_DATA_BLOCK[0], 128)
TRANSMIT_DATA ( OUTGOING_DATA_BLOCK[0], J112_DATA_PORT,
  BYTES_TO_SEND)
  IF DATA_POINTER > TEST_MSG_2_LENGTH THEN EXIT FI
OD

```

END ROUTE_OUT

```

!*****!
GLOBAL
!*****!
MAIN PROCEDURE

ENTRY

    ! PRESET VALUES FOR SOURCE_ADDRESS !
    SOURCE_ADDRESS_ONE := %01
    SOURCE_ADDRESS_TWO := %20
    SOURCE_ADDRESS_THREE := %1D
    SOURCE_ADDRESS_FOUR := %E0

    ! PRESET VALUES FOR DESTINATION_ADDRESS !
    DESTINATION_ADDRESS_ONE := %01
    DESTINATION_ADDRESS_TWO := %20
    DESTINATION_ADDRESS_THREE := %1D
    DESTINATION_ADDRESS_FOUR := %E0

    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, STARTUP_HDR[0], 47)

    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_75[0], 34)

    VECINT

    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_76[0], 45)

    INIT_ZILOG_TABLES

    SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_77[0], 37)

    ROUTE_OUT

    DO

```

```

SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_78[0], 36)
DATA_PRESENT := 0
ROUTE_IN
IF DATA_PRESENT = 0 THEN EXIT FI
OD

```

END MAIN

END MAIN


```

*****
;PROLOGUE -      MODULE ZILOG_LIB
;DATE ORIGINATED: 29 JUL 83
;DATE LAST MODIFIED:
;
;THIS MODULE IS AN ASSEMBLY LIBRARY PACKAGE BUILT
;TO SUPPORT THE ZILOG HOST SOFTWARE WITH NECESSARY LIBRARY ROUTINES
;NOT CURRENTLY AVAILABLE WITH PLZ/SYS. THE MODULE CURRENTLY
;CONSISTS OF PROCEDURES MOVSEQ, RECSEQ, AND SNDSEQ.
;
;GLOBAL MOVSEQ RECSEQ SNDSEQ TRNMIT
;
;EJECT
;PROCEDURE      MOVSEQ          MOVE A SEQUENCE OF BYTES IN MEMORY
;
;THE PURPOSE OF THIS PROCEDURE IS TO MOVE A SEQUENCE
;OF BYTES FROM ONE LOCATION IN MEMORY TO ANOTHER.
;
;INPUT -        THIS PROCEDURE EXPECTS THREE VALUES: THE SOURCE (FROM)
;MEMORY ADDRESS, THE DESTINATION (TO) MEMORY ADDRESS, AND
;THE NUMBER OF BYTES TO BE TRANSFERED. ALL THREE PARAMETERS
;ARE OF WORD LENGTH.
;
;PROCESSING -   THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;WILL BE OFFSET FROM THIS BASE LOCATION.
;THE NEXT SECTION RETRIEVES THE THREE INPUT
;PARAMETERS AND LOADS THEM INTO THE BC, DE, AND HL REG SETS.
;THE LDIR COMMAND WILL MOVE THE CONTENTS OF THE
;LOCATION INDICATED BY HL TO THE LOCATION INDICATED BY DE.
;IT WILL INCREMENT HL AND DE, AND DECREMENT BC. THIS
;PROCEDURE WILL CONTINUE UNTIL BC IS EQUAL TO 0.
;

```

```
; ;  
; ;  
; ;  
    AFTER COMPLETION OF THE LDIR, IX IS RESTORED, THE  
RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE  
STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING  
MODULE IS EXECUTED.  
; ;
```

!OUTPUT - NONE.

```

;INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;
;COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
;PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
;OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
;PLZ USER GUIDE, SECTION 7 FOR DETAILS.
;

```

NOTES -

1. THE NUMBER OF BYTES THAT CAN BE TRANSFERRED BY THIS PROCEDURE IS LIMITED BY THE REGISTER SET SIZE. ATTEMPTING TO TRANSFER A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED IN 16 BITS WILL HAVE UNPREDICTABLE RESULTS.

```

, *****
; MOVSEQ:

```

```
*****  
; ; PROCEDURE TO MOVE A SEQUENCE OF BYTES  
PUSH IX ; STORE IX FOR RETURN  
MOVSEQ:
```

; STORE IX FOR RETURN

; STORE IX FOR RETURN

: SET IX TO BASE OF STACK

IX.0

ADD IX, SP

```
LD C,(IX+4) ; LD BYTES TO MOVE
```

LD B, (IX+5)

: LD DESTINATION ADDRESS

I.D E. (IX+6)

LD D, (IX+7)

: LD SOURCE ADDRESS

I.D I. (IX+8)

LD H, (IX+9)

LDIR ; MOVE BYTES

POP IX ; RESTORE IX

POP HL ; RECOVER RETURN ADDRESS

POP DE ; DEALLOCATE STACK

POP DE

POP DE

JP (HL) ; RETURN

*EJECT

***** PROCEDURE RECSEQ RECEIVE SEQUENCE OF BYTES *****

;

;; THE PURPOSE OF THIS PROCEDURE IS TO RECEIVE A

;; SEQUENCE OF BYTES FROM AN IDENTIFIED PORT.

;

;; INPUT - THIS PROCEDURE EXPECTS FOUR INPUT VALUES: THE
;; USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;; THE STARTING MEMORY LOCATION OF WHERE TO SEND THE DATA,
;; AND THE NUMBER OF BYTES TO RECEIVE. THE TWO USART PORT
;; ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;; THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.

;

;; PROCESSING - THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;; FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;; THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;; WILL BE OFFSET FROM THIS BASE LOCATION.

;; THE NEXT SECTION RETRIEVES THE FOUR INPUT

;; PARAMETERS AND LOADS THEM INTO THE B, C, D, AND HL REGS.

;; A SYNC LOOP IS NEXT FOR CHECKING READY STATUS. THIS

;;

```

; LOOP SYNCHRONIZES THE CODE WITH THE CHOSEN BAUD RATE. THE
; ACTUAL INPUT CODE FOLLOWS WITH THE APPROPRIATE INCREMENT
; AND DECREMENT OF POINTERS. IF THERE ARE BYTES REMAINING
; TO BE RECEIVED, A LOOP BACK TO THE SYNC LOOP CONTINUES THE
; INPUT PROCESS. OTHERWISE, THE IX REG IS RESTORED, THE
; RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
; STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
; MODULE IS EXECUTED.

```

```

; OUTPUT - NONE.

```

```

; INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
; COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
; PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
; AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
; OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,
; PLZ USER GUIDE, SECTION 7 FOR DETAILS.

```

```

; NOTES - 1. THE NUMBER OF BYTES THAT CAN BE RECEIVED
; BY THIS PROC IS LIMITED BY THE REGISTER SIZE. ATTEMPTING
; TO RECEIVE A NUMBER OF BYTES GREATER THAN WHAT CAN BE STORED
; IN 8 BITS WILL HAVE UNPREDICTABLE RESULTS.

```

```

; *****
; RECSEQ: ; PROCEDURE TO RECEIVE SEQUENCE OF BYTES
; PUSH IX ; STORE IX FOR RETURN

```

```

LD IX,0 ; SET IX TO BASE OF STACK
ADD IX,SP

```

```

LD B,(IX+4) ; LD BYTES TO RECEIVE

```

```

LD L,(IX+6) ; LD ADDRESS FOR RECEIPT OF DATA
LD H,(IX+7)

```

```

LD D,(IX+8)      ; LD DATA PORT ADDRESS (CONDAT)
                  ; SINCE USING BITS ONLY THE LOWER
                  ; ORDER BITS ARE USED IN THE
                  ; STATEMENTS ABOVE.

LD C,(IX+10)      ; LD COMAND PORT ADDRESS (CONCMD)

RECLP1           ; WAIT UNTIL READY TO RECEIVE

IN A,(C)
BIT 1,A
JR Z,RECLP1

LD C,D
LD A,(HL)
IN A,(C)          ; LD DATA PORT ADDRESS
                  ; LD DATA ADDRESS
                  ; RECEIVE BYTE

LD (HL),A
INC HL
LD C,(IX+10)
DJNZ RECLP1       ; STORE DATA
                  ; ADVANCE DATA ADDRESS POINTER
                  ; LD COMMAND PORT ADDRESS
                  ; IF NMBR BYTES LEFT > 0, RECEIVE ANOTHER
                  ; ELSE CONTINUE
POP IX
POP HL            ; RESTORE IX
                  ; RECOVER RETURN ADDRESS

POP DE
POP DE
POP DE
POP DE           ; DEALLOCATE STACK

JP (HL)          ; RETURN

```

```

;*****
;*****
;EJECT
;*****

```

```

;PROCEDURE      SNDSEQ      SEND SEQUENCE OF BYTES
;
;      THE PURPOSE OF THIS PROCEDURE IS TO SEND A
;      SEQUENCE OF BYTES TO AN IDENTIFIED PORT.
;
;INPUT -      THIS PROCEDURE EXPECTS FOUR INPUT VALUES: THE
;      USART COMMAND PORT ADDRESS, THE USART DATA PORT ADDRESS,
;      THE STARTING MEMORY LOCATION OF THE DATA TO BE SENT,
;      AND THE NUMBER OF BYTES TO RECEIVE. THE TWO USART PORT
;      ADDRESSES AND THE NUMBER OF BYTES ARE BYTE SIZE WHILE
;      THE MEMORY ADDRESS PARAMETER IS A FULL WORD IN LENGTH.
;
;PROCESSING - THE PROCEDURE BEGINS WITH THE SAVE OF THE IX REG
;      FOR NORMALIZATION AT THE RETURN. THE PUSH ESTABLISHES
;      THE BASE LOCATION FOR THE STACK. ALL INPUT PARAMETERS
;      WILL BE OFFSET FROM THIS BASE LOCATION.
;      THE NEXT SECTION RETRIEVES THE FOUR INPUT
;      PARAMETERS AND LOADS THEM INTO THE B, C, D, AND HL REGS.
;      A SYNC LOOP IS NEXT FOR CHECKING READY STATUS. THIS
;      LOOP SYNCHRONIZES THE CODE WITH THE CHOSEN BAUD RATE. THE
;      ACTUAL OUTPUT CODE FOLLOWS WITH THE APPROPRIATE INCREMENT
;      AND DECREMENT OF POINTERS. IF THERE ARE BYTES REMAINING
;      TO BE SENT, A LOOP BACK TO THE SYNC LOOP CONTINUES THE
;      OUTPUT PROCESS. OTHERWISE, THE IX REG IS RESTORED, THE
;      RETURN ADDRESS IS RECOVERED, THE INPUT PARAMETERS ON THE
;      STACK DEALLOCATED, AND A RETURN JUMP TO THE CALLING
;      MODULE IS EXECUTED.
;
;OUTPUT -      NONE.
;
;INTERFACE - THE INPUT TO THIS PROCEDURE IS OBTAINED VIA STACK
;      COMMUNICATION WITH THE CALLING PLZ MODULE. THE INPUT
;      PARAMETERS ARE LOADED INTO THE STACK WITH A PUSH COMMAND
;      AND ARE RETRIEVED WITH THE USE OF A BASE ADDRESS PLUS AN
;      OFFSET. REFERENCE ZILOG PRODUCT DOCUMENT 03-3096-01,

```

• • •

● ● ●

— — —

SNDSEQ:

PUSH IX

LD IX,0
ADD IX,SP

LD B, (IX+4)

LD L, (IX+6)
LD H, (IX+7)

LD D, (IX+8)

LD C, (IX+10)

SNDLP0

SNDLPI

LD C,D
IN A,(C)

```

RES 7,A
CP 13H
JR NZ, SNDLP3

SNDLP2
LD C,(IX+10)
IN A,(C)
BIT 1,A
JR Z,SNDLP2

LD C,D
IN A,(C)
RES 7,A
CP 11H
JR NZ,SNDLP2

SNDLP3
LD C,D
LD A,(HL)
OUT (C),A

INC HL
LD C,(IX+10)
DJNZ SNDLP0

POP IX
POP HL

POP DE
POP DE
POP DE
POP DE

JP (HL)

; RESET THE EIGHTH BIT
; 13H = ASCII CHARACTER FOR XOFF

; LD COMMAND PORT ADDRESS
; TEST FOR RECEPTION OF A CHARACTER

; LD DATA PORT ADDRESS
; TEST FOR RECEPTION OF XON CHARACTER
; RESET THE EIGHTH BIT
; 11H = ASCII CHARACTER FOR XON

; LD DATA PORT ADDRESS
; LD DATA ADDRESS
; SEND BYTE

; ADVANCE DATA ADDRESS POINTER
; LD COMMAND PORT ADDRESS
; IF NMBR BYTES LEFT > 0, SEND ANOTHER
; ELSE CONTINUE

; RESTORE IX
; RECOVER RETURN ADDRESS

; DEALLOCATE STACK

; RETURN

```

```

;*****

```



```

**EJECT*****TRANSMIT PROCEDURE*****
;PROCEDURE      TRNMIT          THE PURPOSE OF THIS PROCEDURE IS TO TRANSMIT A BYTE
;                OF DATA TO THE UNID.
;
;INPUT - THIS PROCEDURE EXPECTS TWO INPUT PARAMETERS: THE DATA PORT ADDRESS
;        OF THE USART SUPPORTING THE CHANNEL OVER WHICH THE DATA IS TO BE
;        SENT AND THE ADDRESS OF THE DATA. THESE ADDRESSES ARE IN THE GLOBAL
;        VARIABLES OUTADD AND DATADD.
;
;PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE IX REGISTER FOR
;             NORMALIZATION AT THE RETURN. THE NEXT SECTION CONVERTS THE DATA
;             PORT ADDRESS TO THE COMMAND PORT ADDRESS AND LOADS THE ADDRESS INTO
;             THE C-REGISTER. THE DATA IS THEN SENT TO THE UNID. FINALLY, THE IX
;             REGISTER IS RESTORED, THE RETURN ADDRESS RECOVERED,
;             AND A RETURN TO THE CALLING MODULE EXECUTED.
;
;OUTPUT - A COMMAND WORD IS OUTPUT TO THE APPROPRIATE (AS IDENTIFIED BY
;         THE INPUT PARAMETER-OUTADD) USART WHICH CONTAINS A SET TRANSMIT
;         ENABLE BIT.
;
;INTERFACE - THE INPUT TO THIS PROCEDURE IS VIA A GLOBALLY DEFINED PARAMETER
;            FOUND IN ZILOG_APPLICATION MODULE.
;
;NOTES - NONE.
;*****
;TRNMIT:
;           ; PROCEDURE TO TRANSMIT A BLOCK OF DATA
;           ; OUTADD & DATADD LOCATED IN ZILOG_APPLICATION
;           EXTERNAL OUTADD DATADD
;
;           PUSH IX           ; STORE IX REGISTER FOR RETURN
;
;           LD A,(OUTADD)     ; LOAD DATA PORT ADDRESS

```

```

LD C,1          ; CONVERT TO COMMAND PORT ADDRESS
ADD A,C
LD C,A          ; OUTGOING COMMAND PORT ADDRESS

TRNLP1:         ; IS USART READY TO TRANSMIT?
BIT 0,A
JR Z,TRNLP1     ; IF NOT CONTINUE WAITING

LD A,(OUTADD)   ; LOAD DATA PORT ADDRESS
LD C,A
LD DE,(DATADD)  ; LOAD THE DATA
LD A,(DE)

OUT (C),A       ; OUTPUT A BYTE TO UNID

POP IX          ; RESTORE IX REGISTER
POP HL          ; RECOVER RETURN ADDRESS

JP (HL)         ; RETURN TO CALLING PROGRAM

```

```

;*****

```



```

*****
;EJECT
*****
;PROCEDURE      INTUS0      INITIALIZE USART-0
;
;      THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE 8251
;      USART-0 ON THE SIB BOARD.
;
;INPUT - NONE.
;
;PROCESSING - A SEQUENCE OF COMMANDS TO THE USART AND ITS ASSOCIATED
;              COUNTER TIMER CIRCUIT (CTC) ARE REQUIRED FOR INITIALIZATION.
;              THIS PROCEDURE SENDS A USART COMMAND WORD FOLLOWED BY ONE MODE
;              WORD. THE CTC IS THEN ADDRESSED AND A MODE WORD FOLLOWED BY
;              A BAUD RATE PRESCALER CONSTANT IS OUTPUT. THESE ADDRESSES AND
;              THE DATA TO BE OUTPUT ARE PRESET IN A TABLE THAT IS IDENTIFIED
;              BY THE INPUT PARAMETER.
;
;OUTPUT -      THE USART RECEIVES ONE COMMAND WORD FOLLOWED BY ONE
;              MODE COMMAND. THE CTC ASSOCIATED WITH THE USART RECEIVES
;              A COMMAND WORD FOLLOWED BY A BAUD RATE CONSTANT.
;
;INTERFACE -   THIS PROCEDURE IS CALLED BY PROCEDURE-VECINT.
;
;NOTES - 1. INFORMATION CONCERNING THE 8251 USART CAN BE OBTAINED IN:
;          INTEL COMPONENT DATA CATALOG 1981
;          STARTING ON PAGE 8-43
;
*****
;INTUS0:
;          LD B,3
;          LD A,0
;          OUT (ZCOMD0),A
;          DJNZ INTLP1
;          ; SUBROUTINE TO INITIALIZE USART-0
;          ; SEND OUT THREE ZEROS TO RESET USART
;
INTLP1:

```

```

LD A,RESET          ; RESET THE USART
OUT (ZCMD0),A

LD A,MODE           ; SEND OUT MODE COMMAND TO USART
OUT (ZCMD0),A

LD A,COMREG         ; SEND OUT COMMAND TO USART
OUT (ZCMD0),A

IN A,(ZDATA0)       ; CLEAR THE RECEIVE BUFFER

LD A,CTCMOD         ; SEND OUT COUNTER MODE TO CTC
OUT (ZCTC00),A

LD A,RATE19         ; SEND OUT COUNTER VALUE TO CTC
OUT (ZCTC00),A

LD A,VCINT0         ; SETUP TO INTERRUPT WHEN THE RECEIVER
OUT (ZCTC01),A      ; ON THE USART HAS A CHARACTER

LD A,INTCW          ;
OUT (ZCTC01),A

LD A,1              ; INTERRUPT ON EVERY CHARACTER
OUT (ZCTC01),A      ; FOR THE SELECTED USART

RET                 ; RETURN

```

```

;*****
;PROCEDURE INTUS1 INITIALIZE USART-1
;
; THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE 8251
; USART-1 ON THE SIB BOARD.
;

```

```

;INPUT - NONE.
;
;PROCESSING - A SEQUENCE OF COMMANDS TO THE USART AND ITS ASSOCIATED
; COUNTER TIMER CIRCUIT (CTC) ARE REQUIRED FOR INITIALIZATION.
; PROCEDURE INTUS1 SENDS A USART COMMAND WORD FOLLOWED BY A
; MODE WORD. THE CTC IS THEN ADDRESSED AND A MODE WORD FOLLOWED BY
; A BAUD RATE PRESCALER CONSTANT IS OUTPUT. THESE ADDRESSES AND
; THE DATA TO BE OUTPUT ARE PRESET IN A TABLE THAT IS IDENTIFIED
; BY THE INPUT PARAMETER.
;
;OUTPUT - THE USART RECEIVES ONE COMMAND WORD FOLLOWED BY ONE MODE
; COMMAND. THE CTC ASSOCIATED WITH THE USART RECEIVES A COMMAND
; WORD FOLLOWED BY A BAUD RATE CONSTANT.
;
;INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-VECINT.
;
;NOTES - 1. INFORMATION CONCERNING THE 8251 USART CAN BE OBTAINED IN:
; INTEL COMPONENT DATA CATALOG
; STARTING ON PAGE 8-43
;*****
;***** LD B,3 ; SUBROUTINE TO INITIALIZE USART-1
; LD A,0 ; SEND OUT THREE ZEROS TO RESET USART
; OUT (ZCOMD1),A
; DJNZ INTLP2
;
; LD A,RESET ; RESET THE USART
; OUT (ZCOMD1),A
;
; LD A,MODE ; SEND OUT MODE COMMAND TO USART
; OUT (ZCOMD1),A
;
; LD A,COMREG ; SEND OUT COMMAND TO USART
; OUT (ZCOMD1),A

```

```

IN A,(ZDATA1) ; CLEAR THE RECEIVER BUFFER
LD A,CTCMOD ; SEND OUT COUNTER MODE TO CTC
OUT (ZCTC10),A
LD A,RATE19 ; SENT OUT COUNTER VALUE TO CTC
OUT (ZCTC10),A
LD A,VCINT1 ; SETUP TO INTERRUPT WHEN THE RECEIVER
OUT (ZCTC11),A ; ON THE USART HAS A CHARACTER
LD A,INTCW
OUT (ZCTC11),A
LD A,1 ; INTERRUPT ON EVERY CHARACTER
OUT (ZCTC11),A ; FOR THE SELECTED USART
RET ; RETURN
;DEFINES

;NOTE: THE 19.2 KBAUD USED BELOW
;FOR THE CTC TIME CONSTANT PRESCALER
;IS DEPENDENT ON THE SPEED OF THE ZILOG.

USART0 EQU 8CH ; USART 0 INITIALIZATION DATA LIST
ZDATA0 EQU 8DH ; DATA PORT ADDRESS
ZCMD0 EQU 80H ; COMMAND PORT ADDRESS
ZCTC00 EQU 84H ; ASSOCIATED CTC-0 CHANNEL-0 ADDRESS
ZCTC01 EQU 84H ; ASSOCIATED CTC-1 CHANNEL-0 ADDRESS

USART1 EQU 8EH ; USART 1 INITIALIZATION DATA LIST
ZDATA1 EQU 8FH ; DATA PORT ADDRESS
ZCMD1 EQU 80H ; COMMAND PORT ADDRESS
ZCTC10 EQU 85H ; ASSOCIATED CTC-0 CHANNEL-0 ADDRESS
ZCTC11 EQU 85H ; ASSOCIATED CTC-1 CHANNEL-1 ADDRESS

```



```

RESET      EQU      01000000B

; RESET FOR USART-0
; BIT 0 - 0: DISABLE TRANSMIT
; 1 - 0: FORCE NOT(DTR) HIGH
; 2 - 0: DISABLE RECEIVE
; 3 - 0: ASYNCH - NORMAL OPERATION
; 4 - 0: DISABLE ERROR RESET
; 5 - 0: FORCE NOT(RTS) HIGH
; 6 - 1: INTERNAL RESET
; 7 - 0: DISABLE HUNT MODE
;

COMREG     EQU      00110111B

; PROGRAM USART COMMAND REGISTER
; BIT 0 - 1: ENABLE TRANSMIT
; 1 - 1: FORCE NOT(DTR) LOW
; 2 - 1: ENABLE RECEIVE
; 3 - 0: ASYNCH - NORMAL
; 4 - 1: RESET ALL ERROR FLAGS
; 5 - 1: FORCE NOT(RTS) LOW
; 6 - 0: |
; 7 - 0: | NORMAL OPERATION
;

MODE       EQU      010011110B

; PROGRAM USART MODE REGISTER
; BIT 0 - 0: |
; 1 - 1: | ASYNCHRONOUS X16
; 2 - 1: |
; 3 - 1: | 8-BITS PER CHARACTER
; 4 - 0: DISABLE PARITY
; 5 - 0: PARITY (ODD OR EVEN) - DISABLED
; 6 - 1: |
; 7 - 0: | 1-STOP BIT
;

CTCMOD     EQU      01010111B

; CTC CHANNEL CONTROL WORD
; BIT 0 - 1: CONTROL WORD
; 1 - 1: SOFTWARE RESET
; 2 - 1: TIME CONSTANT NEXT WORD

```

```

; 3 - 0: AUTOMATIC TRIGGER
; 4 - 1: SELECT (+) EDGE
; 5 - 0: PRESCALER
; 6 - 1: COUNTER MODE
; 7 - 0: DISABLE INTERRUPTS
; CTC TIME CNST PRESCALER FOR 19.2 KBAUD

RATE19 EQU 2

SIBOFF EQU 01000001B
; CTC CHANNEL CONTROL WORD
; BIT 0 - 1: CONTROL WORD
; 1 - 0: RESET - CONTINUED OPERATION
; 2 - 0: NO TIME CONSTANT NEXT WORD
; 3 - 0: AUTOMATIC TRIGGER
; 4 - 0: SELECT (-) EDGE
; 5 - 0: PRESCALER
; 6 - 1: COUNTER MODE
; 7 - 0: DISABLE INTERRUPTS

INTCW EQU 11010111B
; CHANNEL CONTROL WORD FOR INTERRUPTS
; BIT 0 - 1: ENABLE TRANSMIT
; 1 - 1: FORCE NOT(DTR) LOW
; 2 - 1: ENABLE RECEIVE
; 3 - 0: ASYNCH - NORMAL OPERATION
; 4 - 1: RESET ALL ERROR FLAGS
; 5 - 0: FORCE NOT(RTS) HIGH
; 6 - 1: INTERNAL RESET
; 7 - 1: ENTER HUNT MODE

VCINT0 EQU 40H
VCADD0 EQU 1340H
VCINT1 EQU 42H
VCADD1 EQU 1342H
; INTERRUPT PAGE ADDRESS FOR USART-0
; INTERRUPT TABLE ADDRESS FOR USART-0
; INTERRUPT PAGE ADDRESS FOR USART-1
; INTERRUPT TABLE ADDRESS FOR USART-1

;*****
;EJECT
;*****

```

```

;PROCEDURE      ZRTR01      I/O RECEIVE INTERRUPT CONTROLLER
;
;      THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
;      CHANNEL-1 RECEIVE INTERRUPTS.
;
;INPUT -      THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
;              TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE.  ZL01NS IS
;              THE NEXT SERVICE POSITION; ZL01NE IS THE NEXT EMPTY POSITION;
;              AND ZL01SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
;              ZL01TB.
;
;PROCESSING -  THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
;              PROGRAM'S REGISTERS.
;              THE BYTE IS LOADED INTO THE TABLE WITH LOCATION POSITIONS MODIFIED
;              FOR WRAP-AROUND IF NECESSARY.  FINALLY, THE INTERRUPTED
;              PROGRAM'S REGISTERS ARE RESTORED, INTERRUPTS ENABLED, AND A
;              RETURN FROM INTERRUPT EXECUTED.
;
;OUTPUT -      THE BYTE RECEIVED IS LOADED INTO THE ZL01TB AND
;              THE ZL01NE POSITION IS UPDATED TO REFLECT THE BYTE
;              INSERTION.
;
;INTERFACE -   THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
;              BY THE Z80-CTC.  AS AN INTERRUPT IS IDENTIFIED, THE Z80-CTC
;              DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
;              (IOVCTB).  THIS OFFSET POSITION CONTAINS THE ADDRESS OF
;              THE CORRECT I/O INTERRUPT CONTROLLER.
;
;NOTES -      NONE.
;
;*****
;*****      ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
ZRTR01:
      EXTERNAL ZL01TB ZL01NE ZL01SZ

```

```

EX AF,AF'      ; SAVE REGISTERS OF INTERRUPTED PROGRAM
EXX
PUSH IX
PUSH IY

IN A,(ZDATA0)  ; INPUT THE BYTE

LD DE,ZL01TB   ; SET HL TO NEXT EMPTY TABLE LOCATION
LD HL,(ZL01NE)
ADD HL,DE

LD (HL),A      ; LD BYTE INTO EMPTY TABLE LOCATION

LD HL,(ZL01NE) ; LD EMPTY LOCATION POINTER AND
INC HL         ; INCREMENT EMPTY LOCATION POINTER
LD (ZL01NE),HL ; LD TABLE SIZE FOR CHECK
LD DE,(ZL01SZ) ; IF AT AND OF TABLE, RESET TO LOCATION ZERO
SBC HL,DE
JR NZ,ZR1J10
LD HL,0
LD (ZL01NE),HL

ZR1J10
EX AF,AF'      ; RESTORE CALLING PROGRAMS REGISTERS
EXX
POP IY
POP IX

EI             ; ENABLE INTERRUPTS

RETI          ; RETURN

*****
;EJECT
*****
;PROCEDURE      ZRTR02      I/O RECEIVE INTERRUPT CONTROLLER

```

```

;
; THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL
; CHANNEL-2 RECEIVE INTERRUPTS.
;
; INPUT - THIS PROCEDURE USES THE THREE EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO LOAD THE RECEIVED BYTE. ZL02NS IS
; THE NEXT SERVICE POSITION; ZL02NE IS THE NEXT EMPTY POSITION;
; AND ZL02SZ IS THE TOTAL NUMBER OF BYTE POSITIONS IN TABLE
; ZL02TB.
;
; PROCESSING - THE PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
; PROGRAM'S REGISTERS. THE BYTE IS LOADED INTO
; THE BUFFER AND THE BUFFER LOCATION POSITIONS MODIFIED
; FOR WRAP-AROUND IF NECESSARY. FINALLY, THE INTERRUPTED
; PROGRAM'S REGISTERS ARE RESTORED, AND A RETURN FROM INTERRUPT EXECUTED.
;
; OUTPUT - THE BYTE RECEIVED IS LOADED INTO THE ZL02TB AND
; THE ZL02NE POSITION IS UPDATED TO REFLECT THE BYTE
; INSERTION.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE Z80-CTC. AS AN INTERRUPT IS IDENTIFIED, THE Z80-CTC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
; ***** ; PROCEDURE TO HANDLE RECEIVE I/O INTERRUPTS
ZRTR02:
EXTERNAL ZL02TB ZL02NE ZL02SZ
EX AF,AF' ; SAVE REGISTERS OF INTERRUPTED PROGRAM
EXX

```

```

PUSH IX
PUSH IY

IN A, (ZDATA1)      ; INPUT THE BYTE

LD DE, ZL02TB
LD HL, (ZL02NE)
ADD HL, DE

LD (HL), A          ; LD BYTE INTO EMPTY TABLE LOCATION

LD HL, (ZL02NE)
INC HL
LD (ZL02NE), HL
LD DE, (ZL02SZ)
SBC HL, DE
JR NZ, ZR2J10
LD HL, 0
LD (ZL02NE), HL

ZR2J10
EX AF, AF'          ; RESTORE CALLING PROGRAMS RESISTERS
EXX
POP IY
POP IX

EI                  ; ENABLE INTERRUPTS

RETI                ; RETURN

;*****
;EJECT
;*****
;PROCEDURE      ZRTRN      PROCEDURE TO HANDLE TRANSMIT I/O INTERRUPTS
;
;      THE PURPOSE OF THIS PROCEDURE IS TO SERVICE LOCAL

```

```

; CHANNEL TRANSMIT INTERRUPTS.
;
; INPUT - THIS PROCEDURE USES THE TWO EXTERNALLY DEFINED VALUES
; TO DETERMINE WHERE TO SEND THE BYTE. DATADD IS
; THE DATA SOURCE ADDRESS, AND OUTADD IS THE I/O PORT ADDRESS.
;
; PROCESSING - THIS PROCEDURE BEGINS WITH A SAVE OF THE INTERRUPTED
; PROGRAM'S REGISTERS. THE DATA PORT ADDRESS IS CONVERTED
; TO THE STATUS PORT ADDRESS AND THE PROCEDURE WAITS FOR A
; TRANSMIT READY INDICATION. THE STATUS PORT IS CONVERTED
; BACK TO THE DATA PORT AND THE BYTE IS OUTPUT. THE PROCEDURE
; WAITS AGAIN FOR A READY CONDITION AND THEN RESETS THE
; TRANSMIT ENABLE CONDITION TO GET OUT OF INTERRUPT. FINALLY,
; A RETURN FROM INTERRUPT IS EXECUTED.
;
; OUTPUT - A BYTE OF DATA IS OUTPUT ON THE DATA LINE.
;
; INTERFACE - THIS PROCEDURE IS CALLED VIA INTERRUPT ACTION PROCESSED
; BY THE Z80-CTC. AS AN INTERRUPT IS IDENTIFIED, THE Z80-CTC
; DEVELOPS AN ADDRESS OFFSET INTO THE I/O VECTOR TABLE
; (IOVCTB). THIS OFFSET POSITION CONTAINS THE ADDRESS OF
; THE CORRECT I/O INTERRUPT CONTROLLER.
;
; NOTES - NONE.
;
; *****
; ZRTRN: ; PROCEDURE TO HANDLE TRANSMIT I/O INTERRUPTS *****

EXTERNAL DATADD OUTADD

LD A, (OUTADD) ; LD PORT ADDRESS
LD C, +1 ; CONVERT TO STATUS PORT ADDRESS
ADD A, C
LD C, A

```

```

ZTRNL1      IN A,(C)      ; WAIT UNTIL READY TO TRANSMIT
            BIT 0,A
            JR Z,ZTRNL1

            LD A,-1      ; CONVERT TO DATA PORT ADDRESS
            ADD A,C
            LD C,A
            LD DE,(DATADD) ; LD DATA ADDRESS
            LD A,(DE)

            OUT (C),A      ; OUTPUT A BYTE

            LD A,+1      ; CONVERT TO STATUS PORT ADDRESS
            ADD A,C
            LD C,A
            IN A,(C)      ; WAIT UNTIL READY TO TRANSMIT
            BIT 0,A
            JR Z,ZTRNL2

ZTRNL2      LD A,+2      ; CONVERT TO COMMAND PORT ADDRESS
            ADD A,C
            LD C,A
            IN A,(C)      ; RESET TRANSMIT ENABLE
            RES 0,A
            OUT (C),A

            EI            ; ENABLE INTERRUPTS
            RETI          ; RETURN

```

```

;*****

```



```

*****
*****
***** ZILOG_TABLES      OPERATING SYSTEM TABLES      DATE ORIGINATED: 29 JUL 83
*****
*****
***** DATE LAST MODIFIED:

```

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE ZILOG OPERATING SYSTEM WITH THE TABLES REQUIRED FOR DATA PROCESSING.

THIS MODULE CONSISTS PRIMARILY OF TABLE DEFINITIONS WITH SUBSEQUENT PROCESSING LIMITED TO THE INITIALIZATION OF THE DEFINED TABLES VIA THE PROCEDURE INIT_ZILOG_TABLES.

NOTES: 1. ALL TABLES ARE 128-BYTES IN LENGTH.

2. TABLE LENGTHS CORRESPOND TO INPUT TABLES OF SERVICING UNID.

3. ERROR_TABLE SUBDIVIDED AS:

```

00 = ACCUMULATED ROUTE_IN ERRORS.
01 = ACCUMULATED ROUTE_OUT ERRORS.
02 = INCOMING CONTROL_CODE ERROR.
03 = OUTGOING CONTROL_CODE ERROR.
04 = INCOMING NETWORK_CODE ERROR.
05 = OUTGOING NETWORK_CODE ERROR.
06 = INCOMING HOST_CODE ERROR.
07 = OUTGOING HOST_CODE ERROR.
08 = INCOMING PORT_CODE ERROR.
09 = OUTGOING PORT_CODE ERROR.
10 = NOT USED.

```

...

```

*****
***** 127 = NOT USED.
*****
***** ZILOG_TABLES MODULE
*****

```

```

CONSTANT
DATA_SIZE := 128 ! DATA TO UNID IS IN 128-BYTE BLOCKS !
PACKETS_IN_TABLE := 25
DATA_TABLE_SIZE := DATA_SIZE * PACKETS_IN_TABLE

```

GLOBAL

```

ZZL01NS      INTEGER
ZZL01NE      INTEGER
ZZL01SZ      INTEGER

```

ZZL02NS
ZZL02NE
ZZL02SZ

INCOMING_DATA_BLOCK_ARRAY	[DATA_SIZE BYTE]
OUTGOING_DATA_BLOCK_ARRAY	[DATA_SIZE BYTE]

GLOBAL

```
*****
**      INIT_ZILOG_TABLES    PROCEDURE TO INITIALIZE ZILOG TABLES
**
PROCURE
```

```
PROCEDURE
INIT_ZILOG_TABLES
PROCEDURE TO INITIALIZE ZILOG TABLES
```

THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE DATA TABLES USED BY THE ZILOG OPERATING SYSTEM.

INPUT - NONE.

PROCESSING - THE PROCEDURE INITIALIZES THE FOUR TABLES USED BY THE ZILOG HOST WHEN INTERFACING WITH THE UNID. BY SETTING THE NEXT-BYTE-TO-BE-SERVICED AND THE NEXT-EMPTY-BYTE POINTERS TO ZERO, AND BY SETTING THE TABLE SIZE TO A MULTIPLE OF PACKET SIZE.

OUTPUT - THE TABLE POINTERS AS NOTED UNDER PROCESSING ARE MODIFIED.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-ZILOG_MAIN.

NOTES - 1. ALL LOCAL TABLES ARE 128-BYTES IN LENGTH.

2. ALL TABLES ARE COMPATIBLE WITH SERVICING UNID.

INIT_ZILOG_TABLES PROCEDURE

LOCAL

IX WORD

ENTRY

! INITIALIZE MEMORY TABLE INFORMATION
xxxxns - NEXT BYTE TO BE SERVICED
xxxxne - NEXT EMPTY BYTE
xxxxsz - SIZE OF TABLE
!

! INITIALIZE ZILOG INPUT TABLE-1 !
ZL01ns := 0
ZL01ne := 0
ZL01sz := DATA_TABLE_SIZE

! INITIALIZE ZILOG INPUT TABLE-2 !
ZL02ns := 0
ZL02ne := 0
ZL02sz := DATA_TABLE_SIZE

IX := 0 ! INITIALIZE ERROR_TABLE TO ZEROS !
DO ! INITIALIZE DATA_BLOCK TABLES TO ZEROS !
ERROR_TABLE [IX] := 0

```

INCOMING_DATA_BLOCK [IX] := 0
OUTGOING_DATA_BLOCK [IX] := 0
IX += 1
IF IX = ERROR_TABLE_SIZE THEN EXIT FI
OD

```

```

END INIT_ZILOG_TABLES

```

```

!*****!

```

```

END ZILOG_TABLES

```

```

!*****!

```

```

!*****!

```

```

*****
MODULE -      MODULE ZILOG_TRANSPORT      DATE ORIGINATED: 29 JUL 83
                                           DATE LAST MODIFIED:
*****

```

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE ZILOG_HOST WITH THE TRANSPORT LAYER PROTOCOL. THE ZILOG_HOST IS REQUIRED TO INTERFACE WITH THE UNID FOR BOTH SENDING AND RECEIVING DATA.

THIS MODULE CONSISTS OF THE REQUIRED PROCEDURES FOR THE TRANSPORT LAYER PROTOCOL USING DATAGRAM SERVICE OPTION. THESE PROCEDURES ARE AS FOLLOWS: DATAGRAM_VERSION, DATAGRAM_IHL, DATAGRAM_TYPE_OF_SERVICE, DATAGRAM_TOTAL_LENGTH, DATAGRAM_IDENTIFICATION, DATAGRAM_FLAGS, DATAGRAM_FRAGMENT_OFFSET, DATAGRAM_TIME_TO_LIVE, DATAGRAM_PROTOCOL, DATAGRAM_HEADER_CHECKSUM, DATAGRAM_SOURCE_ADDRESS, DATAGRAM_DESTINATION_ADDRESS, DATAGRAM_SECURITY, DATAGRAM_S_FIELD, DATAGRAM_C_FIELD, DATAGRAM_H_FIELD, DATAGRAM_TCC_FIELD, DATAGRAM_IHF_PADDING, DATAGRAM_SOURCE_PORT, DATAGRAM_DESTINATION_PORT, DATAGRAM_SEQUENCE_NUMBER, DATAGRAM_ACKNOWLEDGEMENT_NUMBER, DATAGRAM_DATA_OFFSET, DATAGRAM_RESERVED, DATAGRAM_CONTROL_BITS, DATAGRAM_WINDOW, DATAGRAM_CHECKSUM, DATAGRAM_URGENT_POINTER, DATAGRAM_OPTION, DATAGRAM_TCP_PADDING, BUILD_TRANSPORT_HEADER, AND EXTRACT_TRANSPORT_HEADER.

```

*****
ZILOG_TRANSPORT MODULE
*****

```

TYPE

PBYTE ^BYTE

CONSTANT

```

! FOLLOWING VALUES ARE PARTICULAR TO THE ZILOG MCZ 1/25 SYSTEM !
PRINTER_DATA_PORT := %90      ! PRINTER DATA PORT ADDRESS !
PRINTER_COMMAND_PORT := %91    ! PRINTER COMMAND PORT ADDRESS !
H19_COMMAND_PORT := %DF        ! H19 CONSOLE COMMAND PORT ADDRESS !

```

```

H19_DATA_PORT := %DE      ! H19 CONSOLE DATA PORT ADDRESS !
DATA_SIZE := 128          ! DATAGRAM WILL BE 128-BYTES LONG !

```

GLOBAL

```

VERSION      BYTE
IHL          BYTE
TYPE_OF_SERVICE  BYTE
TOTAL_LENGTH_HIGH  BYTE
TOTAL_LENGTH_LOW  BYTE
IDENTIFICATION_HIGH  BYTE
IDENTIFICATION_LOW  BYTE
FLAGS        BYTE
FRAGMENT_OFFSET_HIGH  BYTE
FRAGMENT_OFFSET_LOW  BYTE
TIME_TO_LIVE  BYTE
PROTOCOL      BYTE
HEADER_CHECKSUM_HIGH  BYTE
HEADER_CHECKSUM_LOW  BYTE
SECURITY_HIGH  BYTE
SECURITY_LOW  BYTE
S_FIELD_HIGH  BYTE
S_FIELD_LOW  BYTE
C_FIELD_HIGH  BYTE
C_FIELD_LOW  BYTE
H_FIELD_HIGH  BYTE
H_FIELD_LOW  BYTE
TCC_FIELD_ONE  BYTE
TCC_FIELD_TWO  BYTE
TCC_FIELD_THREE  BYTE
IHF_PADDING  BYTE
SOURCE_PORT_HIGH  BYTE
SOURCE_PORT_LOW  BYTE
DESTINATION_PORT_HIGH  BYTE

```

```

DESTINATION_PORT_LOW    BYTE
SEQUENCE_NUMBER_ONE     BYTE
SEQUENCE_NUMBER_TWO     BYTE
SEQUENCE_NUMBER_THREE   BYTE
SEQUENCE_NUMBER_FOUR    BYTE
ACKNOWLEDGEMENT_NUMBER_ONE  BYTE
ACKNOWLEDGEMENT_NUMBER_TWO  BYTE
ACKNOWLEDGEMENT_NUMBER_THREE  BYTE
ACKNOWLEDGEMENT_NUMBER_FOUR  BYTE
      DATA_OFFSET        BYTE
      RESERVED_HIGH       BYTE
      RESERVED_LOW        BYTE
      CONTROL_BITS        BYTE
      WINDOW_HIGH         BYTE
      WINDOW_LOW          BYTE
      CHECKSUM_HIGH       BYTE
      CHECKSUM_LOW        BYTE
      URGENT_POINTER_HIGH  BYTE
      URGENT_POINTER_LOW  BYTE
      OPTION              BYTE
      TCP_PADDING         BYTE

```

```
EXTERNAL      ! LOCATED IN ZILOG_LIB !
```

```

SNDSEQ PROCEDURE (CMDPRT BYTE,DATPRT BYTE,BYTADD PBYTE,NUMBYT BYTE)
RECSEQ PROCEDURE (CMDPRT BYTE,DATPRT BYTE,BYTADD PBYTE,NUMBYT BYTE)
MOVSEQ PROCEDURE (SRCADD PBYTE,DTDADD PBYTE,NUMBYT BYTE)

```

```
EXTERNAL      ! LOCATED IN ZILOG_TABLES !
```

```

INCOMING_DATA_BLOCK_ARRAY [DATA_SIZE BYTE]
OUTGOING_DATA_BLOCK_ARRAY [DATA_SIZE BYTE]

```

```
EXTERNAL      ! LOCATED IN ZILOG_APPLICATION !
```

SOURCE_ADDRESS_ONE BYTE
 SOURCE_ADDRESS_TWO BYTE
 SOURCE_ADDRESS_THREE BYTE
 SOURCE_ADDRESS_FOUR BYTE
 DESTINATION_ADDRESS_ONE BYTE
 DESTINATION_ADDRESS_TWO BYTE
 DESTINATION_ADDRESS_THREE BYTE
 DESTINATION_ADDRESS_FOUR BYTE

INTERNAL ! INTERNAL VARIABLES USED IN THIS MODULE !

! TEST POINTS TO AID IN DEBUGGING !

TP_5	ARRAY	[*BYTE]	:=	'%R<P_5:	EXTRACTING INCOMING DATAGRAM VERSION NUMBER'
TP_6	ARRAY	[*BYTE]	:=	'%R<P_6:	INSERTING OUTGOING DATAGRAM VERSION NUMBER'
TP_7	ARRAY	[*BYTE]	:=	'%R<P_7:	EXTRACTING INCOMING DATAGRAM INTERNET HEADER LENGTH'
TP_8	ARRAY	[*BYTE]	:=	'%R<P_8:	INSERTING OUTGOING DATAGRAM INTERNET HEADER LENGTH'
TP_9	ARRAY	[*BYTE]	:=	'%R<P_9:	EXTRACTING INCOMING DATAGRAM TYPE OF SERVICE'
TP_10	ARRAY	[*BYTE]	:=	'%R<P_10:	INSERTING OUTGOING DATAGRAM TYPE OF SERVICE'
TP_11	ARRAY	[*BYTE]	:=	'%R<P_11:	EXTRACTING INCOMING DATAGRAM TOTAL LENGTH'
TP_12	ARRAY	[*BYTE]	:=	'%R<P_12:	INSERTING OUTGOING DATAGRAM TOTAL LENGTH'
TP_13	ARRAY	[*BYTE]	:=	'%R<P_13:	EXTRACTING INCOMING DATAGRAM IDENTIFICATION'
TP_14	ARRAY	[*BYTE]	:=	'%R<P_14:	INSERTING OUTGOING DATAGRAM IDENTIFICATION'
TP_15	ARRAY	[*BYTE]	:=	'%R<P_15:	EXTRACTING INCOMING DATAGRAM FLAG INFORMATION'
TP_16	ARRAY	[*BYTE]	:=	'%R<P_16:	INSERTING OUTGOING DATAGRAM FLAG INFORMATION'
TP_17	ARRAY	[*BYTE]	:=	'%R<P_17:	EXTRACTING INCOMING DATAGRAM FRAGMENT OFFSET'
TP_18	ARRAY	[*BYTE]	:=	'%R<P_18:	INSERTING OUTGOING DATAGRAM FRAGMENT OFFSET'
TP_19	ARRAY	[*BYTE]	:=	'%R<P_19:	EXTRACTING INCOMING DATAGRAM TIME TO LIVE'
TP_20	ARRAY	[*BYTE]	:=	'%R<P_20:	INSERTING OUTGOING DATAGRAM TIME TO LIVE'
TP_21	ARRAY	[*BYTE]	:=	'%R<P_21:	EXTRACTING INCOMING DATAGRAM PROTOCOL'
TP_22	ARRAY	[*BYTE]	:=	'%R<P_22:	INSERTING OUTGOING DATAGRAM PROTOCOL'
TP_23	ARRAY	[*BYTE]	:=	'%R<P_23:	EXTRACTING INCOMING DATAGRAM HEADER CHECKSUM'
TP_24	ARRAY	[*BYTE]	:=	'%R<P_24:	INSERTING OUTGOING DATAGRAM HEADER CHECKSUM'
TP_25	ARRAY	[*BYTE]	:=	'%R<P_25:	EXTRACTING INCOMING DATAGRAM SOURCE ADDRESS'
TP_26	ARRAY	[*BYTE]	:=	'%R<P_26:	INSERTING OUTGOING DATAGRAM SOURCE ADDRESS'

TP_27	ARRAY	[*BYTE]	:=	'R<P_27:	EXTRACTING INCOMING DATAGRAM DESTINATION ADDRESS'
TP_28	ARRAY	[*BYTE]	:=	'R<P_28:	INSERTING OUTGOING DATAGRAM DESTINATION ADDRESS'
TP_29	ARRAY	[*BYTE]	:=	'R<P_29:	EXTRACTING INCOMING DATAGRAM SECURITY OPTION'
TP_30	ARRAY	[*BYTE]	:=	'R<P_30:	INSERTING OUTGOING DATAGRAM SECURITY OPTION'
TP_31	ARRAY	[*BYTE]	:=	'R<P_31:	EXTRACTING INCOMING DATAGRAM SECURITY LEVEL'
TP_32	ARRAY	[*BYTE]	:=	'R<P_32:	INSERTING OUTGOING DATAGRAM SECURITY LEVEL'
TP_33	ARRAY	[*BYTE]	:=	'R<P_33:	EXTRACTING INCOMING DATAGRAM COMPARTMENT INFORMATION'
TP_34	ARRAY	[*BYTE]	:=	'R<P_34:	INSERTING OUTGOING DATAGRAM COMPARTMENT INFORMATION'
TP_35	ARRAY	[*BYTE]	:=	'R<P_35:	EXTRACTING INCOMING DATAGRAM RESTRICTION INFORMATION'
TP_36	ARRAY	[*BYTE]	:=	'R<P_36:	INSERTING OUTGOING DATAGRAM RESTRICTION INFORMATION'
TP_37	ARRAY	[*BYTE]	:=	'R<P_37:	EXTRACTING INCOMING DATAGRAM GROUP INFORMATION'
TP_38	ARRAY	[*BYTE]	:=	'R<P_38:	INSERTING OUTGOING DATAGRAM GROUP INFORMATION'
TP_39	ARRAY	[*BYTE]	:=	'R<P_39:	PADDING IHF PORTION OF DATAGRAM HEADER'
TP_40	ARRAY	[*BYTE]	:=	'R<P_40:	EXTRACTING INCOMING DATAGRAM SOURCE PORT'
TP_41	ARRAY	[*BYTE]	:=	'R<P_41:	INSERTING OUTGOING DATAGRAM SOURCE PORT'
TP_42	ARRAY	[*BYTE]	:=	'R<P_42:	EXTRACTING INCOMING DATAGRAM DESTINATION PORT'
TP_43	ARRAY	[*BYTE]	:=	'R<P_43:	INSERTING OUTGOING DATAGRAM DESTINATION PORT'
TP_44	ARRAY	[*BYTE]	:=	'R<P_44:	EXTRACTING INCOMING DATAGRAM SEQUENCE NUMBER'
TP_45	ARRAY	[*BYTE]	:=	'R<P_45:	INSERTING OUTGOING DATAGRAM SEQUENCE NUMBER'
TP_46	ARRAY	[*BYTE]	:=	'R<P_46:	EXTRACTING INCOMING DATAGRAM ACKNOWLEDGEMENT NUMBER'
TP_47	ARRAY	[*BYTE]	:=	'R<P_47:	INSERTING OUTGOING DATAGRAM ACKNOWLEDGEMENT NUMBER'
TP_48	ARRAY	[*BYTE]	:=	'R<P_48:	EXTRACTING INCOMING DATAGRAM DATA OFFSET'
TP_49	ARRAY	[*BYTE]	:=	'R<P_49:	INSERTING OUTGOING DATAGRAM DATA OFFSET'
TP_50	ARRAY	[*BYTE]	:=	'R<P_50:	EXTRACTING INCOMING DATAGRAM RESERVED INFORMATION'
TP_51	ARRAY	[*BYTE]	:=	'R<P_51:	INSERTING OUTGOING DATAGRAM RESERVED INFORMATION'
TP_52	ARRAY	[*BYTE]	:=	'R<P_52:	EXTRACTING INCOMING DATAGRAM CONTROL INFORMATION'
TP_53	ARRAY	[*BYTE]	:=	'R<P_53:	INSERTING OUTGOING DATAGRAM CONTROL INFORMATION'
TP_54	ARRAY	[*BYTE]	:=	'R<P_54:	EXTRACTING INCOMING DATAGRAM WINDOW INFORMATION'
TP_55	ARRAY	[*BYTE]	:=	'R<P_55:	INSERTING OUTGOING DATAGRAM WINDOW INFORMATION'
TP_56	ARRAY	[*BYTE]	:=	'R<P_56:	EXTRACTING INCOMING DATAGRAM CHECKSUM VALUE'
TP_57	ARRAY	[*BYTE]	:=	'R<P_57:	INSERTING OUTGOING DATAGRAM CHECKSUM VALUE'
TP_58	ARRAY	[*BYTE]	:=	'R<P_58:	EXTRACTING INCOMING DATAGRAM URGENT POINTER INFORMATION'
TP_59	ARRAY	[*BYTE]	:=	'R<P_59:	INSERTING OUTGOING DATAGRAM URGENT POINTER INFORMATION'
TP_60	ARRAY	[*BYTE]	:=	'R<P_60:	EXTRACTING INCOMING DATAGRAM OPTION (IF ANY)'
TP_61	ARRAY	[*BYTE]	:=	'R<P_61:	INSERTING OUTGOING DATAGRAM OPTION (IF ANY)'

```

TP_62_ARRAY [*BYTE] := '%R&LTP_62: PADDING TCP PORTION OF DATAGRAM HEADER'
!*****!
!***** DATAGRAM_VERSION DETERMINES THE DATAGRAM VERSION NUMBER *****!
PROCEDURE
    THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE VERSION
    NUMBER OF THE DATAGRAM.

INPUT - THIS PROCEDURE EXPECTS A TWO CHARACTER ASCII VALUE INDICATING WHETHER
        THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR
             OUTGOING. IF OUTGOING THEN THE VERSION NUMBER IS INSERTED INTO THE
             DATAGRAM. IF INCOMING THEN THE VERSION NUMBER IS EXTRACTED FROM THE
             DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE WILL RETURN THE DATAGRAM VERSION NUMBER.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS A DATAGRAM
        VERSION NUMBER AND RETURNS A DATAGRAM VERSION NUMBER.
        2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION
           AS THE SMALLEST TYPE. THEREFORE, THE VERSION NUMBER WAS MADE
           INTO A 1-BYTE VALUE WITH THE UPPER 4-BITS REPRESENTING THE
           VERSION NUMBER. WHEN THE "OR" FUNCTION IS PERFORMED, THE VERSION
           NUMBER IS PLACED INTO THE FIRST 4-BITS OF THE DATA_BLOCK ARRAY.
           THE NEXT 4-BITS WILL BE FILLED LATER WITH THE IHL VALUE.

*****!
INTERNAL
*****!

    DATAGRAM_VERSION PROCEDURE (TABLE WORD)

```

```

ENTRY
    IF TABLE
    CASE 'IN'
    THEN
        ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_5[0],51)!
        VERSION := INCOMING_DATA_BLOCK[0] AND %F0
    CASE 'OT'
    THEN
        ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_6[0],50)!
        OUTGOING_DATA_BLOCK[0] := OUTGOING_DATA_BLOCK[0] OR VERSION
    FI ! END IF TABLE CONDITION !

    END DATAGRAM_VERSION

!*****!
!*****!
PROCEDURE   DATAGRAM_IHL           DETERMINES THE DATAGRAM IHL

    THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE INTERNET
    HEADER LENGTH FROM THE DATAGRAM HEADER.

    INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
    DATAGRAM IS INCOMING(IN) OR OUTGOING(OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR
OUTGOING. IF INCOMING THEN THE INTERNET HEADER LENGTH IS EXTRACTED FROM
THE DATAGRAM. IF OUTGOING THEN THE INTERNET HEADER LENGTH IS INSERTED
INTO THE DATAGRAM HEADER.

```

OUTPUT - THIS PROCEDURE RETURNS THE INTERNET HEADER LENGTH OF THE DATAGRAM.
 INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY IHL VALUE.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ALLOWS ONLY BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE, THE IHL WAS MADE INTO A 1-BYTE VALUE WITH THE LOWER 4-BITS REPRESENTING THE IHL NUMBER.
 WHEN THIS IS "OR" INTO THE DATA_BLOCK ARRAY IT IS PLACED INTO THE LOWER 4-BITS OF THE FIRST BYTE.

 INTERNAL
 *****!

DATAGRAM_IHL PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_7[0], 59)!

IHL := INCOMING_DATA_BLOCK[0] AND %0F

CASE 'OT'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_8[0], 58)!

OUTGOING_DATA_BLOCK[0] := OUTGOING_DATA_BLOCK[0] OR IHL

FI ! END IF TABLE CONDITION !

END DATAGRAM_IHL

!*****!

!*****!
PROCEDURE DATAGRAM_TYPE_OF_SERVICE DETERMINES DATAGRAM TYPE OF SERVICE

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE TYPE OF
SERVICE FROM THE DATAGRAM HEADER.

INPUT - THE INPUT TO THIS PROCEDURE IS A TWO CHARACTER ASCII VALUE INDICATING
WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE TYPE OF SERVICE IS EXTRACTED FROM
THE DATAGRAM. IF OUTGOING THEN THE TYPE OF SERVICE IS INSERTED INTO THE
DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE TYPE_OF_SERVICE FOR THE INCOMING DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS OR EXTRACTS A
DUMMY VALUE FOR TYPE_OF_SERVICE.

*****!

INTERNAL

DATAGRAM_TYPE_OF_SERVICE PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_9[0],52)!

```

TYPE_OF_SERVICE := INCOMING_DATA_BLOCK[1]

CASE 'OT'
THEN
  !SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_10[0], 52) !
  OUTGOING_DATA_BLOCK[1] := TYPE_OF_SERVICE

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_TYPE_OF_SERVICE

```

```

!*****

```

```

!*****
PROCEDURE DATAGRAM_TOTAL_LENGTH DETERMINES DATAGRAM TOTAL LENGTH

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE TOTAL LENGTH OF THE DATAGRAM FROM THE DATAGRAM HEADER.

INPUT - THE INPUT TO THIS PROCEDURE IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS OUTGOING OR INCOMING. IF OUTGOING THEN THE DATAGRAM TOTAL LENGTH IS INSERTED INTO THE DATAGRAM. IF INCOMING THEN THE TOTAL LENGTH IS EXTRACTED FROM THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE TOTAL LENGTH OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES : 1. CURRENTLY THIS IS A DUMMY STUB WITH A DUMMY VALUE FOR TOTAL LENGTH.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ALLOWS ONLY BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE, TOTAL_LENGTH WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE UPPER 8-BITS

WHILE THE NEXT 8-BITS REPRESENT THE LOWER 8-BITS OF THE
TOTAL_LENGTH FIELD OF THE DATAGRAM HEADER. THIS SHOULD ALLOW
EASIER MANIPULATION IN THE FUTURE.

*****!

INTERNAL

DATAGRAM_TOTAL_LENGTH PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_11[0], 50) !
TOTAL_LENGTH_HIGH := INCOMING_DATA_BLOCK[2]
TOTAL_LENGTH_LOW := INCOMING_DATA_BLOCK[3]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_12[0], 49) !
OUTGOING_DATA_BLOCK[2] := TOTAL_LENGTH_HIGH
OUTGOING_DATA_BLOCK[3] := TOTAL_LENGTH_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_TOTAL_LENGTH

*****!

*****!

PROCEDURE DATAGRAM_IDENTIFICATION DETERMINES THE DATAGRAM IDENTIFICATION

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE

IDENTIFICATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE IDENTIFICATION IS EXTRACTED FROM THE INCOMING DATAGRAM. IF OUTGOING THEN THE IDENTIFICATION IS INSERTED INTO THE OUTGOING DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM IDENTIFICATION.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS OR RETURNS A DUMMY VALUE FOR IDENTIFICATION.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE, IDENTIFICATION WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE UPPER 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LOWER 8-BITS OF THE IDENTIFICATION FIELD OF THE DATAGRAM HEADER.

INTERNAL

DATAGRAM_IDENTIFICATION PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_13[0],52) !
IDENTIFICATION_HIGH := INCOMING_DATA_BLOCK[4]

IDENTIFICATION_LOW := INCOMING_DATA_BLOCK[5]

**CASE 'OT'
THEN**

```

...:
!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_14[0],51)!
OUTGOING_DATA_BLOCK[4] := IDENTIFICATION_HIGH
OUTGOING_DATA_BLOCK[5] := IDENTIFICATION_LOW

```

FI ! END IF TABLE CONDITION !

END DATAGRAM IDENTIFICATION

```
*****
***** ! *****
***** PROCEDURE *****
***** DATAGRAM *****
***** FLAGS *****
***** DETERMINES *****
***** DATAGRAM *****
***** FLAGGING *****
***** INFORMATION *****
*****
```

THE PURPOSE OF THIS PORCEDURE IS TO EXTRACT/INSERT THE FLAGGING INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING (IN) OR OUTGOING (OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE FLAG INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE FLAG INFORMATION IS INSERTED INTO THE OUTGOING DATAGRAM HEADER.

OUTPUT -- THIS PROCEDURE RETURNS DATAGRAM FLAG INFORMATION.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD TRANSPORT HEADER.

NOTES: 1. CURRENTY THIS IS A DUMMY STUB THAT JUST EITHER RETURNS OR INSERTS DUMMY FLAG INFORMATION.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION

AS THE SMALLEST TYPE. THEREFORE THE FLAG WAS MADE INTO A 1-BYTE
SEGMENT. ONLY THE FIRST 3-BITS REPRESENT THE FLAG INFORMATION.
WHEN THE FLAG IS "OR"ed INTO THE DATA_BLOCK THE FLAG IS INSERTED
INTO THE UPPER 3-BITS OF THE BYTE.

*****!

INTERNAL

DATAGRAM_FLAGS PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_15[0], 54) !

FLAGS := INCOMING_DATA_BLOCK[6] AND %EO

CASE 'OT'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_16[0], 53) !

OUTGOING_DATA_BLOCK[6] := OUTGOING_DATA_BLOCK[6] OR FLAGS

FI ! END IF TABLE CONDITION !

END DATAGRAM_FLAGS

*****!

PROCEDURE DATAGRAM_FRAGMENT_OFFSET DETERMINES DATAGRAM_FRAGMENT_OFFSET

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE FRAGMENT
OFFSET FROM THE DATAGRAM HEADER.

AD-A138 119

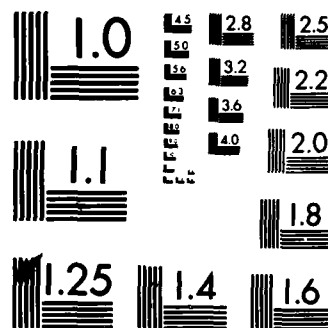
PROTOCOL STANDARDS AND IMPLEMENTATION WITHIN THE
DIGITAL ENGINEERING LABO. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. P W PHISTER
DEC 83 AFIT/GE/EE/83D-58 F/G 17/2

4/4

UNCLASSIFIED

NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE FRAGMENT_OFFSET IS EXTRACTED FROM THE INCOMING DATAGRAM. IF OUTGOING THEN THE FRAGMENT_OFFSET IS INSERTED INTO THE OUTGOING DATAGRAM.

OUTPUT - THIS PROCEDURE RETURNS THE FRAGMENT_OFFSET VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY VALUE FOR FRAGMENT_OFFSET.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE FRAGMENT_OFFSET WAS MADE INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE UPPER 5-BITS AND THE SECOND SEGMENT REPRESENTS THE LOWER 8-BITS OF THE FRAGMENT_OFFSET FIELD OF THE DATAGRAM.

*****!

INTERNAL

DATAGRAM_FRAGMENT_OFFSET PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_17[0], 53) !
FRAGMENT_OFFSET_HIGH := INCOMING_DATA_BLOCK[6] AND \$1F
FRAGMENT_OFFSET_LOW := INCOMING_DATA_BLOCK[7]

```

CASE 'OT'
THEN
  ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_18[0],52) !
  OUTGOING_DATA_BLOCK[6] := OUTGOING_DATA_BLOCK[6] OR
    FRAGMENT_OFFSET_HIGH
  OUTGOING_DATA_BLOCK[7] := FRAGMENT_OFFSET_LOW

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_FRAGMENT_OFFSET

```

```

!*****!

```

```

!*****!
PROCEDURE DATAGRAM_TIME_TO_LIVE DETERMINES THE DATAGRAM TIME TO LIVE

```

```

THIS PROCEDURE EITHER DETERMINES THE TIME_TO_LIVE FOR AN
OUTGOING DATAGRAM OR EXTRACTS THE TIME_TO_LIVE FROM AN INCOMING DATAGRAM.

```

```

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
DATAGRAM IS INCOMING (IN) OR OUTGOING (OT) .

```

```

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE TIME_TO_LIVE IS EXTRACTED FROM THE
DATAGRAM. IF OUTGOING THEN THE TIME_TO_LIVE IS INSERTED INTO THE
DATAGRAM HEADER.

```

```

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM TIME_TO_LIVE.

```

```

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

```

```

NOTES: CURRENTLY THIS IS ONLY A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS
A DUMMY VALUE FOR TIME_TO_LIVE.

```

```

*****!
INTERNAL
    DATAGRAM_TIME_TO_LIVE PROCEDURE (TABLE WORD)

ENTRY

    IF TABLE

        CASE 'IN'
        THEN
            !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_19[0],50)!
            TIME_TO_LIVE := INCOMING_DATA_BLOCK[8]

        CASE 'OT'
        THEN
            !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_20[0],49)!
            OUTGOING_DATA_BLOCK[8] := TIME_TO_LIVE

        FI ! END IF TABLE CONDITION !

    END DATAGRAM_TIME_TO_LIVE

*****!
*****!
PROCEDURE    DATAGRAM_PROTOCOL    DETERMINES DATAGRAM PROTOCOL

    THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE PROTOCOL
    INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
DATAGRAM IS INCOMING (IN) OR OUTGOING (OT) .

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING

```

OR OUTGOING. IF INCOMING THEN THE PROTOCOL IS EXTRACTED FROM THE DATAGRAM
IF OUTGOING THEN THE PROTOCOL IS INSERTED INTO THE OUTGOING DATAGRAM
HEADER.

OUTPUT - THE OUTPUT IS THE DATAGRAM PROTOCOL VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A
DUMMY PROTOCOL VALUE.

*****!

INTERNAL

DATAGRAM_PROTOCOL PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_21[0],46)!

PROTOCOL := INCOMING_DATA_BLOCK[9]

CASE 'OT'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_22[0],45)!

OUTGOING_DATA_BLOCK[9] := PROTOCOL

FI ! END IF TABLE CONDITION !

END DATAGRAM_PROTOCOL

*****!

PROCEDURE DATAGRAM_HEADER_CHECKSUM DETERMINES DATAGRAM HEADER_CHECKSUM

 THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE HEADER
CHECKSUM FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE HEADER_CHECKSUM IS EXTRACTED FROM THE
DATAGRAM. IF OUTGOING THEN THE HEADER_CHECKSUM IS INSERTED INTO THE
OUTGOING DATAGRAM HEADER.

OUTPUT - THE OUTPUT IS THE HEADER_CHECKSUM VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A
 DUMMY HEADER_CHECKSUM VALUE.
2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION
 AS THE SMALLEST TYPE. THEREFORE THE HEADER_CHECKSUM WAS BROKEN INTO
 TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE UPPER 8-BITS
 AND THE SECOND SEGMENT REPRESENTS THE LOWER 8-BITS OF THE HEADER_
CHECKSUM FIELD IN THE DATAGRAM.

INTERNAL

DATAGRAM_HEADER_CHECKSUM PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

```

CASE 'IN'
THEN
  ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_23[0],53) !
  HEADER_CHECKSUM_HIGH := INCOMING_DATA_BLOCK[10]
  HEADER_CHECKSUM_LOW := INCOMING_DATA_BLOCK[11]

CASE 'OT'
THEN
  ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_24[0],52) !
  OUTGOING_DATA_BLOCK[10] := HEADER_CHECKSUM_HIGH
  OUTGOING_DATA_BLOCK[11] := HEADER_CHECKSUM_LOW

```

FI ! END IF TABLE CONDITION !

END DATAGRAM_HEADER_CHECKSUM

!*****! !

!*****! !
 PROCEDURE DATAGRAM_SOURCE_ADDRESS DETERMINES DATAGRAM SOURCE ADDRESS

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE SOURCE
 ADDRESS FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
 DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THE PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR
 OUTGOING. IF INCOMING THEN THE SOURCE ADDRESS IS EXTRACTED FROM THE
 DATAGRAM. IF OUTGOING THEN THE SOURCE ADDRESS IS INSERTED INTO THE
 DATAGRAM HEADER.

OUTPUT - THE OUTPUT IS THE SOURCE ADDRESS OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY VALUE FOR THE SOURCE ADDRESS.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE SOURCE_ADDRESS WAS BROKEN INTO FOUR 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS, THE SECOND SEGMENT REPRESENTS THE SECOND 8-BITS, THE THIRD SEGMENT REPRESENTS THE THIRD 8-BITS, AND THE FOUR SEGMENT REPRESENTS THE LAST 8-BITS OF THE SOURCE_ADDRESS FIELD IN THE DATAGRAM.

INTERNAL

DATAGRAM_SOURCE_ADDRESS PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_25[0], 52) !
SOURCE_ADDRESS_ONE := INCOMING_DATA_BLOCK[12]
SOURCE_ADDRESS_TWO := INCOMING_DATA_BLOCK[13]
SOURCE_ADDRESS_THREE := INCOMING_DATA_BLOCK[14]
SOURCE_ADDRESS_FOUR := INCOMING_DATA_BLOCK[15]

CASE 'OT'

THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_26[0], 51) !
OUTGOING_DATA_BLOCK[12] := SOURCE_ADDRESS_ONE
OUTGOING_DATA_BLOCK[13] := SOURCE_ADDRESS_TWO
OUTGOING_DATA_BLOCK[14] := SOURCE_ADDRESS_THREE
OUTGOING_DATA_BLOCK[15] := SOURCE_ADDRESS_FOUR

FI ! END IF TABLE CONDITION !

END DATAGRAM_SOURCE_ADDRESS

!*****!

!*****
PROCEDURE DATAGRAM_DESTINATION_ADDRESS DETERMINES DATAGRAM DESTINATION

 THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE
 DESTINATION ADDRESS FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM
IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THE PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR
OUTGOING. IF INCOMING THEN THE DESTINATION_ADDRESS IS EXTRACTED FROM THE
DATAGRAM. IF OUTGOING THEN THE DESTINATION_ADDRESS IS INSERTED INTO
THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM DESTINATION_ADDRESS VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ALLOWS ONLY BYTE MANIPULATION
AS THE SMALLEST TYPE. THEREFORE, THE DESTINATION_ADDRESS WAS BROKEN
INTO FOUR 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST
8-BITS, THE SECOND SEGMENT REPRESENTS THE SECOND 8-BITS, THE THIRD
SEGMENT REPRESENTS THE THIRD 8-BITS, AND THE FOURTH SEGMENT
REPRESENTS THE LAST 8-BITS OF THE DESTINATION_ADDRESS FIELD
OF THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_DESTINATION_ADDRESS PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_27[0], 57) !
DESTINATION_ADDRESS_ONE := INCOMING_DATA_BLOCK[16]
DESTINATION_ADDRESS_TWO := INCOMING_DATA_BLOCK[17]
DESTINATION_ADDRESS_THREE := INCOMING_DATA_BLOCK[18]
DESTINATION_ADDRESS_FOUR := INCOMING_DATA_BLOCK[19]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_28[0], 56) !
OUTGOING_DATA_BLOCK[16] := DESTINATION_ADDRESS_ONE
OUTGOING_DATA_BLOCK[17] := DESTINATION_ADDRESS_TWO
OUTGOING_DATA_BLOCK[18] := DESTINATION_ADDRESS_THREE
OUTGOING_DATA_BLOCK[19] := DESTINATION_ADDRESS_FOUR

FI ! END IF TABLE CONDITION !

END DATAGRAM_DESTINATION_ADDRESS

!*****!

!*****!

PROCEDURE DATAGRAM_SECURITY DETERMINES DATAGRAM SECURITY INFORMATION

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE SECURITY
INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE SECURITY INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE SECURITY INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM SECURITY INFORMATION VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY SECURITY INFORMATION VALUE.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE SECURITY CLASSIFICATION WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE UPPER 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LOWER 8-BITS OF THE SECURITY FIELD OF THE DATAGRAM.

*****!
INTERNAL

DATAGRAM_SECURITY PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_29[0], 53) !
SECURITY_HIGH := INCOMING_DATA_BLOCK[20]
SECURITY_LOW := INCOMING_DATA_BLOCK[21]

```

CASE 'OT'
THEN
  ISNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_30[0],52)!
  OUTGOING_DATA_BLOCK[20] := SECURITY_HIGH
  OUTGOING_DATA_BLOCK[21] := SECURITY_LOW

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_SECURITY

```

```

!*****!

```

```

!*****!
PROCEDURE   DATAGRAM_S_FIELD   DETERMINES DATAGRAM SECURITY CLASSIFICATION

```

```

      THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE SECURITY
      CLASSIFICATION FROM THE DATAGRAM HEADER.

```

```

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
        DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

```

```

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE S-FIELD INFORMATION IS EXTRACTED FROM
THE DATAGRAM. IF OUTGOING THEN THE S-FIELD INFORMATION IS INSERTED INTO
THE DATAGRAM HEADER.

```

```

OUTPUT - THE PROCEDURE RETURNS THE DATAGRAM S-FIELD VALUE.

```

```

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BU1 ~ ~ ~ AT_HEADER.

```

```

NOTES: 1. THIS PROCEDURE IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A
        DUMMY S-FIELD VALUE.

```

```

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION
   AS THE SMALLEST TYPE. THEREFORE THE S-FIELD WAS BROKEN INTO TWO
   1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND

```

THE SECOND SEGMENT REPRESENTS THE SECOND 8-BITS OF THE S-FIELD IN
THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_S_FIELD PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_31[0], 52) !
S_FIELD_HIGH := INCOMING_DATA_BLOCK[22]
S_FIELD_LOW := INCOMING_DATA_BLOCK[23]

CASE 'OT'
THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_32[0], 51) !
OUTGOING_DATA_BLOCK[22] := S_FIELD_HIGH
OUTGOING_DATA_BLOCK[23] := S_FIELD_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_S_FIELD

*****!

*****!

PROCEDURE DATAGRAM_C_FIELD DETERMINES DATAGRAM COMPARTMENT INFORMATION

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE
COMPARTMENT INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE C-FIELD INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THE C-FIELD INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM COMPARTMENT VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY C-FIELD VALUE.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE C-FIELD INFORMATION WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE C-FIELD IN THE DATAGRAM HEADER.

INTERNAL

DATAGRAM_C_FIELD PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_33[0],61) !
C_FIELD_HIGH := INCOMING_DATA_BLOCK[24]
C_FIELD_LOW := INCOMING_DATA_BLOCK[25]

```

CASE 'OT'
THEN
  !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_34[0],60)!
  OUTGOING_DATA_BLOCK[24] := C_FIELD_HIGH
  OUTGOING_DATA_BLOCK[25] := C_FIELD_LOW

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_C_FIELD

```

```

!*****!

```

```

!*****!
PROCEDURE DATAGRAM_H_FIELD DETERMINES DATAGRAM RESTRICTION INFORMATION

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT RESTRICTION INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THE PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE H-FIELD INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE H-FIELD INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM RESTRICTION VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS A DUMMY RESTRICTION VALUE.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE H-FIELD WAS BROKEN INTO TWO

1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND
THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE H-FIELD
IN THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_H_FIELD PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_35[0],61) !
H_FIELD_HIGH := INCOMING_DATA_BLOCK[26]
H_FIELD_LOW := INCOMING_DATA_BLOCK[27]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_36[0],60) !
OUTGOING_DATA_BLOCK[26] := H_FIELD_HIGH
OUTGOING_DATA_BLOCK[27] := H_FIELD_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_H_FIELD

*****!

PROCEDURE DATAGRAM_TCC_FIELD DETERMINES DATAGRAM TRANSMISSION CONTROL CODE

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE

TRANSMISSION CONTROL CODE FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE TCC_FIELD IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE TCC_FIELD IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM TRANSMISSION CONTROL CODE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT EITHER INSERTS OR EXTRACTS THE TCC_FIELD INFORMATION FROM THE DATAGRAM HEADER.

2. THE ZILOG 1/25 COMPUTER SYSTEM ONLY ALLOWS BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE TCC_FIELD WAS BROKEN INTO THREE 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS THE SECOND SEGMENT REPRESENTS THE NEXT 8-BITS, AND THE THIRD SEGMENT REPRESENTS THE LAST 8-BITS OF THE TCC_FIELD IN THE DATAGRAM HEADER.

*****!
INTERNAL

DATAGRAM_TCC_FIELD PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_37[0],55)!
TCC_FIELD_ONE := INCOMING_DATA_BLOCK[28]

```

TCC_FIELD_TWO      := INCOMING_DATA_BLOCK[29]
TCC_FIELD_THREE   := INCOMING_DATA_BLOCK[30]

CASE 'OT'
THEN
  ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_38[0], 54) !
  OUTGOING_DATA_BLOCK[28] := TCC_FIELD_ONE
  OUTGOING_DATA_BLOCK[29] := TCC_FIELD_TWO
  OUTGOING_DATA_BLOCK[30] := TCC_FIELD_THREE

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_TCC_FIELD

```

```

!*****!

```

```

!*****!
PROCEDURE   DATAGRAM_IHF_PADDING      PADS THE IHF HEADER WITH ZEROS

```

THE PURPOSE OF THIS PROCEDURE IS TO PAD THE IHF PORTION OF THE DATAGRAM HEADER WITH ZEROS SO THAT THE IHF HEADER WILL END ON AN EVEN 4-BYTE BOUNDARY.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THE IHF PORTION OF THE DATAGRAM HEADER IS PADDED WITH ZEROS UNTIL AN EVEN 4-BYTE BOUNDARY IS REACHED.

OUTPUT - NOTHING.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT PADS A SPECIFIC AMOUNT OF BYTES INTO THE DATAGRAM HEADER TO MAKE IT STOP ON AN EVEN 4-BYTE BOUNDARY.

2. SINCE THE DATAGRAM STARTS WITH BYTE-0 THE 4-BYTE BOUNDARY WILL BE
 BYTE-31.

*****!
 INTERNAL

DATAGRAM_IHF_PADDING PROCEDURE

ENTRY

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_39[0],47)!
 OUTGOING_DATA_BLOCK[31] := IHF_PADDING

END DATAGRAM_IHF_PADDING

*****!

*****!
 PROCEDURE DATAGRAM_SOURCE_PORT DETERMINES THE DATAGRAM SOURCE PORT

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE SOURCE
 PORT FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
 DATAGRAM IN INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR
 OUTGOING. IF INCOMING THEN THE SOURCE_PORT IS EXTRACTED FROM THE
 DATAGRAM. IF OUTGOING THEN THE SOURCE_PORT IS INSERTED INTO THE OUTGOING
 DATAGRAM.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM SOURCE PORT VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS OR EXTRACTS A DUMMY SOURCE_PORT VALUE INTO THE DATAGRAM.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE SOURCE_PORT WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE SOURCE_PORT FIELD OF THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_SOURCE_PORT PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_40[0], 49) !

SOURCE_PORT_HIGH := INCOMING_DATA_BLOCK[32]

SOURCE_PORT_LOW := INCOMING_DATA_BLOCK[33]

CASE 'OT'

THEN

ISNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_41[0], 48) !

OUTGOING_DATA_BLOCK[32] := SOURCE_PORT_HIGH

OUTGOING_DATA_BLOCK[33] := SOURCE_PORT_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_SOURCE_PORT

!*****!

!*****
PROCEDURE DATAGRAM_DESTINATION_PORT DETERMINES DATAGRAM DESTINATION PORT

 THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE
 DESTINATION PORT FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
 DATAGRAM IS INCOMING(IN) OR OUTGOING(OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE DESTINATION_PORT IS EXTRACTED FROM THE
DATAGRAM. IF OUTGOING THEN THE DESTINATION_PORT IS INSERTED INTO THE
DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATAGRAM DESTINATION_PORT VALUE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS OR EXTRACTS A DUMMY
 DESTINATION_PORT VALUE.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION
 AS THE SMALLEST TYPE. THEREFORE DESTINATION_PORT WAS BROKEN INTO TWO
 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND
 THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE DESTINATION_PORT
 IN THE DATAGRAM HEADER.

INTERNAL

DATAGRAM_DESTINATION_PORT PROCEDURE (TABLE WORD)

ENTRY

IF TABLE


```

CASE 'IN'
THEN
    !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_42[0],54) !
    DESTINATION_PORT_HIGH := INCOMING_DATA_BLOCK[34]
    DESTINATION_PORT_LOW  := INCOMING_DATA_BLOCK[35]

CASE 'OT'
THEN
    !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_43[0],53) !
    OUTGOING_DATA_BLOCK[34] := DESTINATION_PORT_HIGH
    OUTGOING_DATA_BLOCK[35] := DESTINATION_PORT_LOW

```

FI ! END IF TABLE CONDITION !

END DATAGRAM_DESTINATION_PORT

!*****!

!*****!

PROCEDURE DATAGRAM_SEQUENCE_NUMBER DETERMINES DATAGRAM SEQUENCE NUMBER

 THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE SEQUENCE
 NUMBER FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
 DATAGRAM IS INCOMING(IN) OR OUTGOING(OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
 OR OUTGOING. IF INCOMING THEN THE SEQUENCE_NUMBER IS EXTRACTED FROM THE
 DATAGRAM. IF OUTGOING THEN THE SEQUENCE_NUMBER IS INSERTED INTO THE
 DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE SEQUENCE_NUMBER OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE SEQUENCE_NUMBER WAS BROKEN INTO FOUR 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS, THE SECOND SEGMENT REPRESENTS THE NEXT 8-BITS, THE THIRD SEGMENT REPRESENTS THE NEXT 8-BITS, AND THE FOURTH SEGMENT REPRESENTS THE LAST 8-BITS OF THE SEQUENCE NUMBER FIELD IN THE DATAGRAM HEADER.

DATAGRAM_SEQUENCE_NUMBER PROCEDURE (TABLE WORD)

IF TABLE

CASE 'IN'

THEN

```

!$NDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_44[0],53)!
SEQUENCE_NUMBER_ONE      := INCOMING_DATA_BLOCK[36]
SEQUENCE_NUMBER_TWO      := INCOMING_DATA_BLOCK[37]
SEQUENCE_NUMBER_THREE    := INCOMING_DATA_BLOCK[38]
SEQUENCE_NUMBER_FOUR     := INCOMING_DATA_BLOCK[39]

```

CASE 'OT'

THEN

```

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_45[0],52)!
OUTGOING_DATA_BLOCK[36] := SEQUENCE_NUMBER_ONE
OUTGOING_DATA_BLOCK[37] := SEQUENCE_NUMBER_TWO
OUTGOING_DATA_BLOCK[38] := SEQUENCE_NUMBER_THREE
OUTGOING_DATA_BLOCK[39] := SEQUENCE_NUMBER_FOUR

```

FI ! END IF TABLE CONDITION !

END DATAGRAM_SEQUENCE_NUMBER

!*****!

!*****!

PROCEDURE DATAGRAM_ACKNOWLEDGEMENT_NUMBER DETERMINES THE DATAGRAM ACK NO.

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE
ACKNOWLEDGEMENT NUMBER FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
OR OUTGOING. IF INCOMING THEN THE ACKNOWLEDGEMENT_NUMBER IS EXTRACTED
FROM THE DATAGRAM. IF OUTGOING THEN THE ACKNOWLEDGEMENT_NUMBER IS
INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE ACKNOWLEDGEMENT_NUMBER OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY
ACKNOWLEDGEMENT_NUMBER VALUE.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION
AS THE SMALLEST TYPE. THEREFORE THE ACKNOWLEDGEMENT_NUMBER WAS
BROKEN INTO FOUR 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE
FIRST 8-BITS, THE SECOND SEGMENT REPRESENTS THE NEXT 8-BITS, THE
THIRD SEGMENT REPRESENTS THE NEXT 8-BITS, AND THE FOURTH SEGMENT
REPRESENTS THE LAST 8-BITS OF THE ACKNOWLEDGEMENT_NUMBER FIELD IN
THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_ACKNOWLEDGEMENT_NUMBER PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

ISNDSEQ (PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_46[0], 60) !
ACKNOWLEDGEMENT_NUMBER_ONE := INCOMING_DATA_BLOCK[40]
ACKNOWLEDGEMENT_NUMBER_TWO := INCOMING_DATA_BLOCK[41]
ACKNOWLEDGEMENT_NUMBER_THREE := INCOMING_DATA_BLOCK[42]
ACKNOWLEDGEMENT_NUMBER_FOUR := INCOMING_DATA_BLOCK[43]

CASE 'OT'
THEN

ISNDSEQ (PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_47[0], 59) !
OUTGOING_DATA_BLOCK[40] := ACKNOWLEDGEMENT_NUMBER_ONE
OUTGOING_DATA_BLOCK[41] := ACKNOWLEDGEMENT_NUMBER_TWO
OUTGOING_DATA_BLOCK[42] := ACKNOWLEDGEMENT_NUMBER_THREE
OUTGOING_DATA_BLOCK[43] := ACKNOWLEDGEMENT_NUMBER_FOUR

FI ! END IF TABLE CONDITION !

END DATAGRAM_ACKNOWLEDGEMENT_NUMBER

!*****!

!*****!

PROCEDURE DATAGRAM_DATA_OFFSET DETERMINES THE DATAGRAM DATA OFFSET

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE DATA
OFFSET FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE DATA_OFFSET IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE DATA_OFFSET IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE DATA_OFFSET VALUE OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY DATA_OFFSET VALUE.
2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE DATA_OFFSET WAS MADE INTO A 1-BYTE SEGMENT WITH ONLY THE UPPER 4-BITS REPRESENTING THE DATA_OFFSET FIELD IN THE DATAGRAM HEADER.

INTERNAL

DATAGRAM_DATA_OFFSET PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_48[0],49) !
DATA_OFFSET := INCOMING_DATA_BLOCK[44] AND %F0

CASE 'OT'

```

THEN
  !SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_49[0],48) !
  OUTGOING_DATA_BLOCK[44] := OUTGOING_DATA_BLOCK[44] OR
    DATA_OFFSET

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_DATA_OFFSET

```

```

!*****!

```

```

!*****!
PROCEDURE DATAGRAM_RESERVED DETERMINES DATAGRAM RESERVED INFORMATION

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE RESERVED INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING (IN) OR OUTGOING (OT) .

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE RESERVED INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE RESERVED INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE RESERVED INFORMATION FROM THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY RESERVED VALUE.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE RESERVED FIELD IN THE DATAGRAM WAS MADE INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 4-BITS (LOWER 4-BITS OF BYTE) AND THE SECOND SEGMENT REPRESENTS

THE REMIANING 2-BITS (UPPER 2-BITS OF SECOND BYTE) OF THE RESERVED
FIELD IN THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_RESERVED PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_50[0],58) !
RESERVED_HIGH := INCOMING_DATA_BLOCK[44] AND %0F
RESERVED_LOW := INCOMING_DATA_BLOCK[45] AND %C0

CASE 'OT'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_51[0],57) !
OUTGOING_DATA_BLOCK[44] := OUTGOING_DATA_BLOCK[44] OR
RESERVED_HIGH
OUTGOING_DATA_BLOCK[45] := OUTGOING_DATA_BLOCK[45] OR
RESERVED_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_RESERVED

*****!

*****!

PROCEDURE DATAGRAM_CONTROL_BITS DETERMINES DATAGRAM CONTROL INFO

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT CONTROL INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE CONTROL INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE CONTROL INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE CONTROL INFORMATION FROM THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY CONTROL FIELD FROM THE DATAGRAM.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE CONTROL FIELD WAS MADE INTO A 1-BYTE SEGMENT. THE LAST 6-BITS OF THE BYTE REPRESENTS THE CONTROL FIELD IN THE DATAGRAM HEADER.

*****!
INTERNAL

DATAGRAM_CONTROL_BITS PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_52[0],57)!
CONTROL_BITS := INCOMING_DATA_BLOCK[45] AND %3F


```

CASE 'OT'
THEN
  !SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_53[0], 56) !
  OUTGOING_DATA_BLOCK[45] := OUTGOING_DATA_BLOCK[45] OR
    CONTROL_BITS

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_CONTROL_BITS

```

```

!*****!

```

```

!*****!
PROCEDURE DATAGRAM_WINDOW DETERMINES DATAGRAM WINDOW INFO

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE WINDOW INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE WINDOW INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE WINDOW INFORMATION IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE WINDOW INFORMATION FROM THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES:
1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY WINDOW VALUE.
 2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE WINDOW FIELD OF THE DATAGRAM

HEADER WAS MADE INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT
REPRESENTS THE FIRST 8-BITS AND THE SECOND SEGMENT REPRESENTS THE
LAST 8-BITS OF THE WINDOW FIELD IN THE DATAGRAM HEADER.

*****!

INTERNAL

DATAGRAM_WINDOW PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_54[0], 56) !
WINDOW_HIGH := INCOMING_DATA_BLOCK[46]
WINDOW_LOW := INCOMING_DATA_BLOCK[47]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_55[0], 55) !
OUTGOING_DATA_BLOCK[46] := WINDOW_HIGH
OUTGOING_DATA_BLOCK[47] := WINDOW_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_WINDOW

*****!

*****!

PROCEDURE DATAGRAM_CHECKSUM DETERMINES THE DATAGRAM CHECKSUM

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT CHECKSUM

INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES IF THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN THE CHECKSUM INFORMATION IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN THE CHECKSUM VALUE IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE CHECKSUM VALUE OF THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

- NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY CHECKSUM VALUE.
2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION AS THE SMALLEST TYPE. THEREFORE THE CHECKSUM WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE CHECKSUM FIELD IN THE DATAGRAM HEADER.

INTERNAL

DATAGRAM_CHECKSUM PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'

THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_56[0],52) !
CHECKSUM_HIGH := INCOMING_DATA_BLOCK[48]

```

CHECKSUM_LOW := INCOMING_DATA_BLOCK[49]

CASE 'OT'
THEN
  !SNDSEQ(PRINTER_COMMAND_PORT, PRINTER_DATA_PORT, TP_57[0], 51) !
  OUTGOING_DATA_BLOCK[48] := CHECKSUM_HIGH
  OUTGOING_DATA_BLOCK[49] := CHECKSUM_LOW

```

```

FI ! END IF TABLE CONDITION !

```

```

END DATAGRAM_CHECKSUM

```

```

!*****!

```

```

!*****!

```

```

PROCEDURE DATAGRAM_URGENT_POINTER DETERMINES DATAGRAM URGENT INFO

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACT/INSERT THE URGENT
 POINTER INFORMATION FROM THE DATAGRAM HEADER.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE
 DATAGRAM IS INCOMING (IN) OR OUTGOING (OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING
 OR OUTGOING. IF INCOMING THEN THE URGENT_POINTER INFORMATION IS EXTRACTED
 FROM THE DATAGRAM. IF OUTGOING THEN THE URGENT_POINTER INFORMATION IS
 INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS THE URGENT_POINTER INFORMATION.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY
 URGENT_POINTER VALUE.

2. THE ZILOG MCZ 1/25 COMPUTER SYSTEM ONLY ALLOWS FOR BYTE MANIPULATION

AS THE SMALLEST TYPE. THEREFORE THE URGENT_POINTER FIELD WAS BROKEN INTO TWO 1-BYTE SEGMENTS. THE FIRST SEGMENT REPRESENTS THE FIRST 8-BITS AND THE SECOND SEGMENT REPRESENTS THE LAST 8-BITS OF THE URGENT_POINTER FIELD IN THE DATAGRAM HEADER.

INTERNAL*****!

DATAGRAM_URGENT_POINTER PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_58[0],64) !
URGENT_POINTER_HIGH := INCOMING_DATA_BLOCK[50]
URGENT_POINTER_LOW := INCOMING_DATA_BLOCK[51]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_59[0],63) !
OUTGOING_DATA_BLOCK[50] := URGENT_POINTER_HIGH
OUTGOING_DATA_BLOCK[51] := URGENT_POINTER_LOW

FI ! END IF TABLE CONDITION !

END DATAGRAM_URGENT_POINTER

!*****!

!*****
PROCEDURE DATAGRAM_OPTION INSERTS ANY DATAGRAM OPTIONS SELECTED

THE PURPOSE OF THIS PROCEDURE IS TO INSERT OR EXTRACT ANY OPTION DATA USED WITHIN THE DATAGRAM.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING WHETHER THE DATAGRAM IS INCOMING(IN) OR OUTGOING(OT).

PROCESSING - THIS PROCEDURE FIRST DETERMINES WHETHER THE DATAGRAM IS INCOMING OR OUTGOING. IF INCOMING THEN ANY OPTIONS SELECTED IS EXTRACTED FROM THE DATAGRAM. IF OUTGOING THEN ANY OPTIONS SELECTED IS INSERTED INTO THE DATAGRAM HEADER.

OUTPUT - THIS PROCEDURE RETURNS ANY OPTIONS SELECTED FOR THE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. CURRENTLY THIS PROCEDURE IS A DUMMY STUB THAT JUST INSERTS AND EXTRACTS A DUMMY OPTION VALUE.

*****!
INTERNAL

DATAGRAM_OPTION PROCEDURE (TABLE WORD)

ENTRY

IF TABLE

CASE 'IN'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_60[0],53) !
OPTION := INCOMING_DATA_BLOCK[52]

CASE 'OT'
THEN

!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_61[0],52) !

FI ! END IF TABLE CONDITION !

END DATAGRAM OPTION

— * * * * *

```
*****
!*****
PROCEDURE      DATAGRAM_TCP_PADDING      PADS THE TCP PORTION WITH ZEROS
*****
```

THE PURPOSE OF THIS PROCEDURE IS TO PAD THE TCP PORTION OF THE DATAGRAM HEADER WITH ZEROS. THE TCP PORTION OF THE DATAGRAM HEADER MUST END ON AN EVEN 4-BYTE BOUNDARY.

INPUT - NONE.

PROCESSING - THIS PROCEDURE PADS THE TCP PORTION OF THE DATAGRAM HEADER WITH ZEROS SO THAT IT ENDS ON AN EVEN 4-BYTE BOUNDARY.

OUTPUT - NONE.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE-BUILD_TRANSPORT_HEADER.

NOTES: 1. THIS PROCEDURE IS A DUMMY STUB THAT JUST PADS IN 3-BYTES OF ZEROS.

INTERNAL

DATAGRAM TCP PADDING PROCEDURE

ENTRY

```
!SNDSEQ(PRINTER_COMMAND_PORT,PRINTER_DATA_PORT, TP_62[0],47)!
OUTGOING DATA_BLOCK[53] := TCP_PADDING
```

OUTGOING_DATA_BLOCK[54] := TCP_PADDING
OUTGOING_DATA_BLOCK[55] := TCP_PADDING

END DATAGRAM_TCP_PADDING

!*****!

!*****!
PROCEDURE BUILD_TRANSPORT_HEADER BUILD THE IHF+TCP TRANSPORT HEADER

THE PURPOSE OF THIS PROCEDURE IS TO BUILD THE IHF+TCP HEADER
OF THE TRANSPORT LAYER.

INPUT - NONE.

PROCESSING - CURRENTLY THIS PROCEDURE JUST CYCLES THROUGH THE DATAGRAM HEADER
INSERTING ONE FIELD AFTER ANOTHER WITH DUMMY DATA.

OUTPUT - A FULLY FILLED DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE ROUTE_OUT IN
THE ZILOG MAIN DRIVER.

NOTES - 1. THIS PROCEDURE CREATES A DUMMY TRANSPORT HEADER WITH ONLY
THE DESTINATION ADDRESS INSERTED.

2. THE INDIVIDUAL HEADER VALUES WERE GIVEN THE ASCII VALUE OF 2F
WHICH WILL PRINT A "/" IN THE DATAGRAM HEADER. ONLY THE SOURCE AND
DESTINATION WILL CONTAIN VALID DATA.

*****!

GLOBAL

BUILD_TRANSPORT_HEADER PROCEDURE

ENTRY

! THIS SECTION INITIALIZES ALL THE DATAGRAM DATA TO
PRESCRIBED VALUES. AS THE DATAGRAM HEADER IS EXPANDED
THESE VALUES WILL TAKE ON VALID VALUES. !

VERSION := %20

IHL := %0F

TYPE_OF_SERVICE := %2F

TOTAL_LENGTH_HIGH := %2F

TOTAL_LENGTH_LOW := %2F

IDENTIFICATION_HIGH := %2F

IDENTIFICATION_LOW := %2F

FLAGS := %20

FRAGMENT_OFFSET_HIGH := %0F

FRAGMENT_OFFSET_LOW := %2F

TIME_TO_LIVE := %2F

PROTOCOL := %2F

HEADER_CHECKSUM_HIGH := %2F

HEADER_CHECKSUM_LOW := %2F

SECURITY_HIGH := %2F

SECURITY_LOW := %2F

S_FIELD_HIGH := %2F

S_FIELD_LOW := %2F

C_FIELD_HIGH := %2F

C_FIELD_LOW := %2F

H_FIELD_HIGH := %2F

H_FIELD_LOW := %2F

TCC_FIELD_ONE := %2F

TCC_FIELD_TWO := %2F

TCC_FIELD_THREE := %2F

IHF_PADDING := %2F

SOURCE_PORT_HIGH := %2F

SOURCE_PORT_LOW := %2F

DESTINATION_PORT_HIGH := %2F

DESTINATION_PORT_LOW := %2F

SEQUENCE_NUMBER_ONE := %2F

SEQUENCE_NUMBER_TWO := %2F

```

SEQUENCE_NUMBER_THREE := %2F
SEQUENCE_NUMBER_FOUR := %2F
ACKNOWLEDGEMENT_NUMBER_ONE := %2F
ACKNOWLEDGEMENT_NUMBER_TWO := %2F
ACKNOWLEDGEMENT_NUMBER_THREE := %2F
ACKNOWLEDGEMENT_NUMBER_FOUR := %2F
DATA_OFFSET := %2F
RESERVED_HIGH := %0F
RESERVED_LOW := %00
CONTROL_BITS := %2F
WINDOW_HIGH := %2F
WINDOW_LOW := %2F
CHECKSUM_HIGH := %2F
CHECKSUM_LOW := %2F
URGENT_POINTER_HIGH := %2F
URGENT_POINTER_LOW := %2F
OPTION := %2F
TCP_PADDING := %2F

```

! THIS SECTION BUILDS THE IHF PORTION OF THE TRANSPORT HEADER !

```

DATAGRAM_VERSION ('OT')
DATAGRAM_IHL ('OT')
DATAGRAM_TYPE_OF_SERVICE ('OT')
DATAGRAM_TOTAL_LENGTH ('OT')
DATAGRAM_IDENTIFICATION ('OT')
DATAGRAM_FLAGS ('OT')
DATAGRAM_FRAGMENT_OFFSET ('OT')
DATAGRAM_TIME_TO_LIVE ('OT')
DATAGRAM_PROTOCOL ('OT')
DATAGRAM_HEADER_CHECKSUM ('OT')
DATAGRAM_SOURCE_ADDRESS ('OT')
DATAGRAM_DESTINATION_ADDRESS ('OT')
DATAGRAM_SECURITY ('OT')
DATAGRAM_S_FIELD ('OT')

```

```

DATAGRAM_C_FIELD ('OT')
DATAGRAM_H_FIELD ('OT')
DATAGRAM_TCC_FIELD ('OT')

```

```

! PAD THE REST OF THE IHF HEADER WITH ZEROS. !
! IHF HEADER MUST END ON AN EVEN 4-BYTE BOUNDARY !
DATAGRAM_IHF_PADDING

```

```

! INSERT TCP PORTION INTO THE DATAGRAM HEADER !

```

```

DATAGRAM_SOURCE_PORT ('OT')
DATAGRAM_DESTINATION_PORT ('OT')
DATAGRAM_SEQUENCE_NUMBER ('OT')
DATAGRAM_ACKNOWLEDGEMENT_NUMBER ('OT')
DATAGRAM_DATA_OFFSET ('OT')
DATAGRAM_RESERVED ('OT')
DATAGRAM_CONTROL_BITS ('OT')
DATAGRAM_WINDOW ('OT')
DATAGRAM_CHECKSUM ('OT')
DATAGRAM_URGENT_POINTER ('OT')
DATAGRAM_OPTION ('OT')

```

```

! PAD THE TCP HEADER TO AN EVEN 4-BYTE BOUNDARY !
DATAGRAM_TCP_PADDING

```

```

END BUILD_TRANSPORT_HEADER

```

```

!*****!

```

```

!*****!
PROCEDURE EXTRACT_TRANSPORT_HEADER EXTRACTS DATAGRAM FIELDS FROM HEADER

```

THE PURPOSE OF THIS PROCEDURE IS TO EXTRACTS EACH OF THE FIELDS
FROM THE DATAGRAM HEADER.

PROCESSING - CURRENTLY THIS PROCEDURE JUST CYCLES THROUGH AND EXTRACTS EACH OF THE DATAGRAM FIELDS ONE AFTER THE OTHER.

INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE-ROUTE_IN IN THE ZILOG MAIN DRIVER.

EXTRACT_TRANSPORT_HEADER PROCEDURE

! THIS SECTION EXTRACTS THE IHF PORTION OF DATAGRAM HEADER !

```

DATAGRAM_VERSION ('IN')
DATAGRAM_IHL ('IN')
DATAGRAM_TYPE_OF_SERVICE ('IN')
DATAGRAM_TOTAL_LENGTH ('IN')
DATAGRAM_IDENTIFICATION ('IN')
DATAGRAM_FLAGS ('IN')
DATAGRAM_FRAGMENT_OFFSET ('IN')
DATAGRAM_TIME_TO_LIVE ('IN')
DATAGRAM_PROTOCOL ('IN')
DATAGRAM_HEADER_CHECKSUM ('IN')
DATAGRAM_SOURCE_ADDRESS ('IN')
DATAGRAM_DESTINATION_ADDRESS ('IN')
DATAGRAM_SECURITY ('IN')
DATAGRAM_S_FIELD ('IN')

```

DATAGRAM_C_FIELD ('IN')
DATAGRAM_H_FIELD ('IN')
DATAGRAM_TCC_FIELD ('IN')

! THIS SECTION EXTRACTS THE TCP PORTION OF DATAGRAM HEADER !

DATAGRAM_SOURCE_PORT ('IN')
DATAGRAM_DESTINATION_PORT ('IN')
DATAGRAM_SEQUENCE_NUMBER ('IN')
DATAGRAM_ACKNOWLEDGEMENT_NUMBER ('IN')
DATAGRAM_DATA_OFFSET ('IN')
DATAGRAM_RESERVED ('IN')
DATAGRAM_CONTROL_BITS ('IN')
DATAGRAM_WINDOW ('IN')
DATAGRAM_CHECKSUM ('IN')
DATAGRAM_URGENT_POINTER ('IN')
DATAGRAM_OPTION ('IN')

END EXTRACT_TRANSPORT_HEADER

!*****!

END ZILOG_TRANSPORT

!*****!

!*****!

VITA

Captain Paul W. Phister, Jr. was born on 7 August 1950 in Seattle, Washington. In 1968, Capt Phister graduated from Waynesboro Area Senior High School. Upon graduation he entered Hagerstown Junior College, where in 1970 Capt Phister received an Associate of Arts degree in Drafting and Design Technology.

In 1970 he enlisted in the U.S. Army and served as a CRYPTO Repairman (MOS 32F2T) in Bremerhaven Germany until 1973 when he entered the University of Akron. In 1977, Capt Phister received his Bachelor of Science degree in Electrical Engineering. He was commissioned in December 1977 and was assigned to the Foreign Technology Division, AFSC, at Wright-Patterson AFB, Ohio.

In 1979, Capt Phister was assigned to the Air Force Electronic Warfare Center, ESC, at Kelly AFB, Texas. While assigned at Kelly AFB, Capt Phister received a Master of Science degree in Systems Management from St Mary's University in San Antonio, Texas.

Capt Phister entered the Air Force Institute of Technology in June 1982.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/EE/8E/83D-58	2. GOVT ACCESSION NO. AD-138 119	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROTOCOL STANDARDS AND IMPLEMENTATION WITHIN THE DIGITAL ENGINEERING LABORATORY COMPUTER NETWORK (DELNET) USING THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID)		5. TYPE OF REPORT & PERIOD COVERED MASTER OF SCIENCE THESIS JUNE 1982 - DECEMBER 1983
7. AUTHOR(s) PAUL W. PHISTER, JR.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (AFIT) ELECTRICAL ENGINEERING DEPARTMENT (EN) WRIGHT-PATTERSON AFB, OH 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (AFIT) ELECTRICAL ENGINEERING DEPARTMENT (EN) WRIGHT-PATTERSON AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE DECEMBER 1983
		13. NUMBER OF PAGES 621
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES APPROVED FOR RELEASE BY AIR 190-17. Lyn Wolan. 7 Feb 84 Dean for Technical and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
APPLICATION LAYER DELNET	PRESENTATION LAYER RS-449	X.21
COMPUTER NETWORKS ISO REFERENCE MODEL	ROUTING	DATA LINK LAYER X.25
DATAGRAM NETWORK LAYER	RS-232	UNID
DATA LINK LAYER PHYSICAL LAYER	RS-422/423	VIRTUAL CIRCUIT
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) See Reverse		

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

(Block 20 Continued)

Development of the Air Force Institute of Technology's Digital Engineering Laboratory Network (DELNET) was continued with the development of an initial draft of a protocol standard for all seven layers as specified by the International Standards Organization's (ISO) Reference Model for Open Systems Interconnections. This effort centered on the restructuring of the Network Layer to perform Datagram routing and to conform to the developed protocol standards and actual software module development of the upper four protocol layers residing within the DELNET Monitor (Zilog MCZ 1/25 Computer System). Within the guidelines of the ISO Reference Model the Transport Layer was developed utilizing the Internet Header Format (IHF) combined with the Transport Control Protocol (TCP) to create a 128-byte Datagram. Also a limited Application Layer was created to pass the Gettysburg Address through the DELNET. This study formulated a first draft for the DELNET Protocol Standard and designed, implemented, and tested the Network, Transport, and Application Layers to conform to these protocol standards.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

END

FILMED

384

DTIC